

MISTA 2009

Proceedings of the

4th Multidisciplinary International Conference on Scheduling: Theory and Applications

10th – 12th August 2009
Dublin, Ireland

Edited by

Jacek Blazewicz, Poznan University of Technology, Poland

Maciej Drozdowski, Poznan University of Technology, Poland

Graham Kendall, University of Nottingham, UK

Barry McCollum, Queens University Belfast, UK

Published by University of Nottingham, UK

ISSN 2305-249X

ISBN-10: 0-9545821-6-0

ISBN-13: 978-0-9545821-6-6

MISTA 2009 Conference Program Committee

Abdullah, Salwani	Henz, Martin
Aickelin, Uwe	Herrmann, Jeffrey
Alfares, Hesham	Herroelen, Willy
Allahverdi, Ali	Hoogeveen, Han
Artiba, Hakim	Ibaraki, Toshihide
Artigues, Christian	Janiak, Adam
Ayob, Masri	Jost, Vincent
Baptiste, Philippe	Jouglet, Antoine
Bartak, Roman	Kendall, Graham (conference chair)
Bean, James	Kingston, Jeffrey
Beck, Chris	Knust, Sigrid
Bini, Enrico	Kubiak, Wieslaw
Blazewicz, Jacek (conference chair)	Kutanoglu, Erhan
Burke, Edmund	Kwan, Raymond
Brucker, Peter	Lau, Hoong Chuin
Cai, Xiaoqiang	Le Pape, Claude
Carlier, Jacques	Lee, Chung-Yee
Chen, Zhi-Long	Leung, Joseph
Cheng, Tai Chiu Edwin	Levner, Eugene
Chrétienne, Philippe	Løkketangen, Arne
Cucu-Grosjean, Liliana	Mattfeld, Dirk
Dauzère-Pérès, Stéphane	McCollum, Barry (conference chair)
De Causmaecker, Patrick	McMullan, Paul
Coffmann, Ed	Middendorf, Martin
Dell'Amico, Mauro	Möhring, Rolf
Demeulemeester, Erik	Munier-Kordon, Alix
Dowsland, Kath	Norman, Bryan
Dror, Moshe	Nuijten, Wim
Drozdowski, Maciej (conference chair)	Oguz, Ceyda
Erben, Wilhelm	Osman, Ibrahim
Finke, Gerd	Ozcan, Ender
Fisher, Nathan	Pappis, Costas
Fogel, David	Pesch, Erwin
Froncek, Dalibor	Petrovic, Sanja
Gendreau, Michel	Petrovic, Dobrila
Glass, Celia	Pinedo, Mike
Glover, Fred	Portmann, Marie-Claude
Glabot, Bernard	Potvin, Jean-Yves
Guinet, Alain	Prins, Christian
Hao, Jin-Kao	Rayward-Smith, Vic

Ribeiro, Celso
Rudova, Hana
Sagir, Mujgan
Schaerf, Andrea
Slowinski, Roman
Smith, Stephen
Sourd, Francis
T'kindt, Vincent
Thompson, Jonathan
Trick, Michael
Trystram, Denis

van de Velde, Steef
Van Hentenryck, Pascal
Vanden Berghe, Greet
Vazacopoulos, Alkis
Watson, Jean-Paul
Wein, Joel
Woeginger, Gerhard
White, George
Wright, Mike
Zinder, Yakov

MISTA 2005 International Advisory Committee

- Graham Kendall (chair)
- Abdelhakim Artiba, Facultes Universitaires Catholiques de Mons (CREGI - FUCAM), Belgium
- James Bean, University of Michigan, USA
- Jacek Blazewicz, Institute of Computing Science, Poznan University of Technology, Poland
- Peter Brucker, University of Osnabrück, Germany
- Edmund Burke, The University of Nottingham, UK
- Xiaoqiang Cai, The Chinese University of Hong Kong, Hong Kong
- Ed Coffman, Columbia University, USA
- Moshe Dror, The University of Arizona, USA
- David Fogel, Natural Selection Inc., USA
- Michel Gendreau, University of Montreal, Canada
- Fred Glover, Leeds School of Business, University of Colorado, USA
- Bernard Grabot, Laboratoire Génie de Production - ENIT, Tarbes, France
- Toshihide Ibaraki, Kyoto University, Japan
- Claude Le Pape, ILOG, France
- Ibrahim Osman, American University of Beirut, Lebanon
- Michael Pinedo, New York University, USA
- Jean-Yves Potvin, Université de Montréal, Canada
- Michael Trick, Graduate School of Industrial Administration, Carnegie Mellon University, USA
- Stephen Smith, Carnegie Mellon University, USA
- Steef van de Velde, Erasmus University, Netherlands
- George White, University of Ottawa, Canada
- Gerhard Woeginger, University of Twente, Netherlands

Acknowledgements

This conference would not have been possible without the assistance of many people.

We would like to thank the program committee for doing such an excellent job of reviewing the papers. Without them lending their help and expertise, this conference could not have taken place.

The editors would also like to thank the international advisory committee for their helpful advice and comments during the planning and preparation of this conference. Their advice, as always, enables us to ensure that we are meeting the needs of the community.

We appreciate the financial support that we received from Tracsis plc and Taylor and Francis.

We are also grateful for the support we have received from the University of Nottingham, Queen's University, Belfast, EventMAP Ltd. and AMT (Aircraft Management Technologies)

The conference would not have been possible without the help of the administrative support we received from the School of Computer Science at the University of Nottingham. Nick Poxon and Ebru Tasci helped a great deal in the immediate run up to the conference.

Finally, the conference owes a great deal to Debbie Pitchfork. She has worked tirelessly for about 18 months in the build up to the conference and if it were not for Debbie this conference would not have taken place.

Thank you Debbie.

Table of Contents

Plenary Papers

Atkin J.A.D., Burke E.K. and Greenwood J.S. <i>A Comparison of Two Methods for Reducing Take-Off Delay at London Heathrow Airport</i>	12
Dror M. and Steiner G. <i>‘Strong’-‘Weak’ Precedence in Scheduling: Extended Order Implications</i>	28
Kwan R.S.K. <i>Case Studies of Successful Train Crew Scheduling</i>	43

Papers

Serafini P. <i>On some Combinatorial Properties of PESP</i>	59
Kvaratskhelia A. and Lazarev A. <i>Polynomial Algorithm for $1 \mid r_j, p_j = p, pmnt \mid \sum w_i c_j$ Scheduling Problem</i>	68
Bini E. <i>Modeling Preemptive EDF and FP by Integer Variables</i>	77
Kergosien Y., Lenté C. and Billaut J-C. <i>Home Health Care Problem: An Extended Multiple Traveling Salesman Problem</i>	85
Xu Y. and Qu R. <i>A GRASP Approach for the Delay-Constrained Multicast Routing Problem</i>	93
Mönch L. and Almeder C. <i>Ant Colony Optimization Approaches for Scheduling Jobs with Incompatible Families on Parallel Batch Machines</i>	105
Guedes A.C.B. and Ribeiro C.C. <i>A Heuristic for Minimizing Weighted Carry-Over Effects in Round Robin Tournaments</i>	115
Alsheddy A. and Tsang E. <i>Empowerment-based Workforce Scheduling Problem</i>	130
Gunawan A. and Lau H.C. <i>Master Physician Scheduling Problem</i>	145
Al-Betar M.A. and Khader A.T. <i>A Hybrid Harmony Search for University Course Timetabling</i>	157
Jat S.N. and Yang S. <i>A Guided Search Genetic Algorithm for the University Course Timetabling Problem</i>	180
Mönch L., Fowler J.W., Dauzère-Pérès S., Mason S.J. and Rose O. <i>Scheduling Semiconductor Manufacturing Operations: Problems, Solution Techniques, and Future Challenges</i>	192
Osman I.H., Belouadah H., Fleszar K. and Saffar M. <i>Hybrid of the Weighted Minimum Slack and Shortest Processing Time Dispatching Rules for the Total Weighted Tardiness Single Machine Scheduling Problem with Availability Constraints</i>	202

Tanaka S. and Sato S. <i>An Exact Algorithm for the Precedence-Constrained Single-Machine Scheduling Problem</i>	216
Ourari S., Briand C. and Bouzouia B. <i>Minimizing the Number of Tardy Jobs in Single Machine Scheduling using MIP</i>	227
Hanzálek Z. and Šůcha P. <i>Time Symmetry of Project Scheduling with Time Windows and Take-Give Resources</i>	239
Benabid A. and Hanen C. <i>Decomposed Software Pipelining for Cyclic Unitary RCPSP with Precedence Delays</i>	254
Sterna M. and Juraszek J. <i>Scheduling Policy for On-Line Translation Service</i>	266
Guitouni A. and Masri H. <i>A Nonlinear Mixed Integer Program for Search Path Planning Problem (SPPP)</i>	277
Hazir O., Kedad-Sidhoum S. and Chrétienne P. <i>Batching and Scheduling with Tardiness Penalties</i>	291
Shabtay D. and Steiner G. <i>Bicriteria Models to Minimize the Total Weighted Number of Tardy Jobs with Convex Controllable Processing Times and Common Due Date Assignment</i>	301
Dožić S., Kalić M., Babić O. and Čangalović M. <i>Heuristic Approach to the Airline Schedule Disturbances Problem: Multi-Fleet Case</i>	311
Berlińska J. and Drozdowski M. <i>Heuristics for Divisible Loads Scheduling in Systems with Limited Memory</i>	321
Baker K.R. and Trietsch D. <i>Three Heuristic Procedures for the Stochastic Flow Shop Problem</i>	330
Ormsbee L., Lingireddy S. and Chase D. <i>Optimal Pump Scheduling for Water Distribution Systems</i>	341
Khowala K., Fowler J., Keha A. and Balasubramanian H. <i>Single Machine Scheduling with Interfering Job Sets</i>	357
Vakhania N. <i>An Efficient Implicit Enumeration for Scheduling Equal-Length Jobs with Release Times on a Single Processor to Maximize Throughput</i>	366
Simonin G., Darties B., Giroudeau R. and König J-C. <i>Isomorphic Coupled-Task Scheduling Problem with Compatibility Constraints on a Single Processor</i>	378
Vakhania N. <i>Scheduling Jobs with Release Times Preemptively on a Single Machine to Minimize the Number of Late Jobs</i>	389
Sabar N.R., Ayob M. and Kendall G. <i>Solving Examination Timetabling Problems using Honey-Bee Mating Optimization (ETP-HBMO)</i>	399
Burke E.K., Qu R. and Soghier A. <i>Adaptive Selection of Heuristics within a GRASP for Exam Timetabling Problems</i>	409

McCollum B., McMullan P.J., Parkes A.J., Burke E.K. and Abdullah S. <i>An Extended Great Deluge Approach to the Examination Timetabling Problem</i>	424
Qu R., He F. and Burke E.K. <i>Hybridizing Integer Programming Models within an Adaptive Decomposition Approach for Exam Timetabling Problems</i>	435
Pillay N. <i>Evolving Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem</i>	447
Kravchenko S.A. and Werner F. <i>Parallel Machine Problems with Equal Processing Times</i>	458
Aggoune R. and Mati Y. <i>Recent Advances in Solving Two-Job Shop Scheduling Problems: Extensions and Improvements of the Geometric Approach</i>	469
Mokotoff E. <i>Minimizing the Makespan and Total Flow Time on the Permutation Flow Shop Scheduling Problem</i>	479
Barták R. and Skalický T. <i>A local Approach to Automated Correction of Violated Precedence and Resource Constraints in Manually Altered Schedules</i>	507
Duša V. and Barták R. <i>Preemptive Scheduling with Precedences and Alternative Resources</i>	518
Chen S-H., Chen M-C., Chang P-C., Zhang Q. and Chen Y-M. <i>Development of Effective Estimation of Distribution Algorithms for Scheduling Problems</i>	531
Schwiegelshohn U. <i>An Owner-Centric Metric for the Evaluation of Online Job Schedules</i>	557
Pappis C.P., Rachaniotis N.P. and Dasaklis T. <i>A Deterministic Resource Scheduling Model in Epidemic Logistics</i>	570
Detienne B., Dauzère-Pérès S. and Yugma C. <i>Scheduling Inspection Operations Subject to a Fixed Production Schedule</i>	581
Ceschia S. and Schaerf A. <i>Tabu Search Techniques for the Heterogeneous Vehicle Routing Problem with Time Windows and Carrier-Dependent Costs</i>	594
Tran T.H. and Ng K.M. <i>A Water Flow Algorithm for Flexible Flow Shop Scheduling with Limited Intermediate Buffers</i>	606
Papa G. and Korošec P. <i>Metaheuristic Approach to Loading Schedule Problem</i>	616
Herrmann J.W. <i>Generating Cyclic Fair Sequences for Multiple Servers</i>	628
Elvikis D., Hamacher H.W. and T'kindt V. <i>Scheduling Two Interfering Job Sets on Uniform Parallel Machines with Makespan and Cost Functions</i>	645
Nishi T. and Hiranaka Y. <i>Lagrangian Relaxation and Cut Generation for Sequence Dependent Setup Time Flowshop Scheduling Problems</i>	655

Abstracts

Lazarev A.A. and Gafarov E.R. <i>Estimation of Absolute Error for the Resources-Constrained Project Scheduling Problem</i>	671
Lehoux-Lebacque V., Brauner N. and Finke G. <i>Polynomial Complexity of the Cyclic Identical Coupled Task Problem</i>	679
Bilgin B., De Causmaecker P. and Vanden Berghe G. <i>A Hyperheuristic Approach to Belgian Nurse Rostering Problems</i>	683
Maenhout B. and Vanhoucke M. <i>Reactive Nurse Scheduling in Hospitals</i>	690
Haspeslagh S., De Causmaecker P. and Vanden Berghe G. <i>A Multi-Agent System Handling Personnel Shortages in Hospitals</i>	693
Fowler J.W., Gul S., Denton B.T. and Huschka T.R. <i>Bi-Criteria Scheduling of an Outpatient Procedure Center</i>	696
Juraszek J., Pesch E. and Sterna M. <i>Simulated Annealing Method for Maximizing Revenue on Parallel Machines</i>	699
Bożejko W., Uchroński M. and Wodecki M. <i>Scatter Search for a Weighted Tardiness Flow Shop Problem</i>	702
Almeder C., Katzensteiner S. and Hartl R.F. <i>A VNS-Based Solution Approach for a Flexible Flow Shop Problem with Limited Buffer</i>	706
Abdul-Hamid N.H., Kendall G. and Sagir M. <i>Mathematical Modeling for Maximising Gate Receipt Problem</i>	709
Goossens D.R. and Spieksma F.C.R. <i>Soccer Schedules in Europe: An Overview</i>	715
Lewis R. and Thompson J. <i>Round-Robin Sports Scheduling from a Graph Colouring Perspective: A Case Study in Rugby Union Scheduling</i>	718
Santos C.A., Zhang A., Gonzalez M.T., Jain S. and Byde A. <i>Workforce Planning and Scheduling for the HP IT Services Business</i>	720
Beliën J. and Demeulemeester E. <i>Integrated Workforce Staffing and Scheduling: A Case Study at a Major Aircraft Maintenance Company</i>	724
Abdullah S., Turabieh H., McCollum B. and Burke E.K. <i>An investigation of a Genetic Algorithm and Sequential Local Search Approach for the Curriculum-Based Course Timetabling Problem</i>	727
Müller T., Rudová H. and Murray K. <i>Interactive Course Timetabling</i>	732
Parkes A.J., McCollum B., McMullan P. and Burke E.K. <i>Recent Work on Planning and Management of University Teaching Space</i>	737
Günther H-O. <i>An MILP-Based Approach to Short-Term Production Scheduling and Lot Sizing with Major and Minor Setups: A Case Study from the Beverage Industry</i>	741
Oğuz C. and Narin Ö. <i>Solving Berth Allocation Problem with Column Generation</i>	744

Błażewicz J., Machowiak M. and Oğuz C. <i>Approximation Algorithm for Berth and Quay Crane Allocation Problem with Bounds</i>	748
Berghman L. and Leus R. <i>Solving a Dock Assignment Problem as a Three-Stage Flexible Flow-Shop Problem</i>	752
Creemers S., De Reyck B., Lambrecht M. and Leus R. <i>Project Scheduling with Alternative Technologies and Stochastic Activity Durations</i>	756
Hurink J.L., Kok A.L., Paulus J.J. and Schutten J.M.J. <i>Time-Constrained Project Scheduling with Adjacent Resources</i>	761
Lin E.Y.H. <i>On the Effective Approach of Teaching Activity Scheduling in Project Management</i>	764
Weeks K.O., Gao H., Alidaee B. and Rana D. <i>An Empirical Study of Impacts of Production Mix Efficiency, Product Route Efficiency on Operations Performance and Profitability: A Reverse Logistics Approach</i>	765
Zerola M., Barták R., Lauret J. and Šumbera M. <i>Planning Heuristics for Efficient Data Movement on the Grid</i>	768
Nobibon F.T., Herbots J. and Leus R. <i>Order Acceptance and Scheduling in a Single-Machine Environment: Exact and Heuristic Algorithms</i>	772
Pawlak G. and Rucinski M. <i>Single Machine Re-Entrant Jobs Scheduling in the Car Factory Paint Shop</i>	775
McCollum B., Kwan R.S.K. and McMullan P.J. <i>Real World Optimisation through Spin Out Activity</i>	778
Glass C. and Chakhlevitch K. <i>Scheduling in a Micro-Biological Laboratory: Heuristics for Sequencing Food Sample Tests, to Co-ordinate with Production of Perishable Media Materials</i>	782
Özcan E., Misir M. and Burke E.K. <i>A Self-Organising Hyper-heuristic Framework</i>	784
Özcan E. and Burke E.K. <i>Multilevel Search for Choosing Hyper-Heuristics</i>	788
Burke E.K., Curtois T., Hyde M., Kendall G., Ochoa G., Petrovic S. and Vázquez-Rodriguez J.A. <i>HyFlex: A Flexible Framework for the Design and Analysis of Hyper-Heuristics</i> ...	790
Hyde M., Özcan E. and Burke E.K. <i>Multilevel Search for Evolving the Acceptance Criteria of a Hyper-Heuristic</i>	798
Gomes M.C., Barbosa-Póvoa A. and Novais A.Q. <i>Scheduling of Job Shop, Make-to-Order Industries with Recirculation and Assembly: Discrete Versus Continuous Time Models</i>	802
Ciavotta M., Ruiz R. and Minella G. <i>New Results for the Multi-Objective Sequence Dependent Setup Times Flowshop Problem</i>	808
Gather T. and Zimmermann J. <i>Exact Methods for the Resource Levelling Problem</i>	811

Løkketangen A., Haugen K., Lanquepin G. and Olstad A. <i>Using a Set of Fixed Prices when Solving Profit Maximization Capacitated Lot-Size Problems (PCLSP)</i>	821
Abd-Aziz Z. and Kendall G. <i>An Investigation of an Ant-based Hyperheuristic for the Capacitated Vehicle Routing Problem</i>	823
Borgulya I. <i>Solving the Undirected Capacitated Arc Routing Problem with a Memory Based Method</i>	827
Gendreau M., Côté J-F. and Potvin J-Y. <i>An Effective Heuristic for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading and Multiple Stacks</i>	831
Ruiz R. and Naderi B. <i>Variable Neighborhood Descent Methods for the Distributed Permutation Flowshop Problem</i>	834
McCollum B., Kendall G., McMullan P. and McGoldrick J. <i>Decision Support Systems for Tankering within the Airline Industry</i>	837
Bril R.J., Cucu-Grosjean L. and Goossens J. <i>Response Time Analysis in Distributed Real-Time Systems: An Improvement Based on Best-Case Finalization Time Analysis</i>	839
Yomsi P.M. and Sorel Y. <i>Schedulability Analysis for non Necessarily Harmonic Real-Time Systems with Precedence and Strict Periodicity Constraints using the Exact Number of Preemptions and no Idle Time</i>	843
Rudec T., Baumgartner A. and Manger R. <i>Speeding up the Optimal Off-Line Algorithm for Solving the k-server Problem</i>	852

Plenary Papers

A comparison of two methods for reducing take-off delay at London Heathrow airport

Jason A. D. Atkin • Edmund K. Burke • John S Greenwood

Abstract This paper describes recent research into the departure process at London Heathrow, with the primary motivation of reducing the amount of fuel used, improving both the economic and environmental cost. Two proposals are considered here. The first proposal considers the practicality and potential benefits of aiding the controller to improve the take-off sequencing. The second proposal aims to absorb some of the inevitable delay for aircraft at the stands, before the engines are started, but also involves a take-off sequencing aspect. Models for the two take-off sequencing problems are presented in this paper, the second of which includes an additional pushback time (or TSAT) allocation sub-problem which has to be solved subsequently. These models have distinctive differences from the models for the take-off and arrival sequencing problems which are usually considered in the literature, since they take into account necessary constraints imposed due to the control problem (whether a sequence can actually be achieved, and how) in each case. As discussed in this paper, the control problem cannot be ignored by the controllers at Heathrow, so cannot be ignored by any realistic system to aid them in their tasks. Comparative take-off sequencing results are presented for the two systems and the potential benefits from providing decision support to the runway controllers or improved TSAT allocation at the stands are considered. The paper ends with some overall conclusions from the research, showing the large potential benefits of these systems. The TSAT allocation system which is discussed in this paper has been developed for implementation at London Heathrow as one element of the Collaborative Decision Making project.

1 Introduction

This paper considers potential improvements to the departure process at London Heathrow airport in order to reduce the delay for aircraft awaiting take-off, thus reducing the amount of fuel burnt, leading to cost and environmental benefits. Models are presented in this paper for two problems at different stages of the departure system and solution systems are proposed and

Jason A. D. Atkin and Edmund K. Burke
School of Computer Science, The University of Nottingham, Jubilee Campus,
Nottingham, NG8 1BB, UK
E-mail: jaa@cs.nott.ac.uk, ekb@cs.nott.ac.uk

John S. Greenwood
NATS, NATS CTC, 4000 Parkway, Whiteley, Fareham, Hampshire, PO15 7FL, UK

evaluated. The first system considers the possibility of improving the take-off sequencing by providing a decision support system to the controller who performs that task. The second system considers the potential to absorb some of the necessary delay for aircraft at the stand so that the fuel burn is further reduced. It has similarities to the first system in that it must predict take-off sequences and it may indirectly aid the controller to improve the take-off sequence by reducing congestion in the holding area and providing a more appropriate selection of aircraft from which to select. The potential benefits from both of these systems are evaluated and contrasted in this paper.

From the point of view of an automated system to help the controllers in the control tower, the departure process has a number of stages. When granted permission to do so, aircraft push back from their stands, start their engines and enter the taxiways. Once on the taxiways, one or more ground controllers will direct the aircraft to a holding area by the current departure runway, where they queue awaiting their turn to enter the runway and take off. At some point prior to take-off, a take-off sequence must be determined. Under present operating procedures, a runway controller will re-sequence the aircraft within the holding area, attempting to improve the take-off sequence and reduce delay for aircraft.

The first system which is described in this paper has been designed as a decision support system for the runway controller. It attempts to make decisions about the take-off sequence to adopt by considering known controller preferences and operating methods as well as the physical and regulatory constraints upon the sequencing problem. One of the main requirements of such a system is that it must react virtually instantaneously to changes in the situation at the runway, thus results have to be determined within a second. Obviously, there are human-computer-interface (HCI) issues to resolve in order to incorporate such a system for the runway controller, since it is vital (for safety-related reasons) that it does not distract the controller or increase the workload. So far, this research has been useful for providing insight into the role of the runway controller, the constraints upon the departure system and the likely effects of any changes to the way in which the departure system operates.

The second system which is described in this paper has been designed to assign Target Start-At Times (TSATs) to aircraft while they are still at the stands. TSATs define the times at which an aircraft should be given permission to push back from the stands, start their engines and commence their journeys towards the take-off runway. A take-off sequencing element is again involved in this problem, but decision times can be longer. This second system is currently being integrated into Heathrow.

2 Take-off sequencing

The runway forms the bottleneck for the departure system at London Heathrow. Although it is such a popular and busy airport, Heathrow currently has only two runways and, despite the fact that mixed mode operations (using both runways for interleaved take-offs and landings) would be more efficient, usually only one of the runways is available for take-offs at any time of the day - the other being used simultaneously for landings. This restriction is designed to control the noise for residents on the flight paths. Since the runway is normally used purely for take-offs, landings do not need to be considered in the models presented here.

A minimum separation is required between consecutive take-offs to allow wake vortices to dissipate. This separation depends upon the relative weight classes of the two aircraft and is asymmetric. An additional minimum separation must also be met between aircraft which depart along the same or similar departure routes, even if they are not consecutive take-offs (the separations do not obey the triangle inequality), in order to control the rate of flow along each route. This route-based separation can depend upon the time of day (temporary increases may be applied at times) and will change depending upon the runway that is in use and the direction of take-off. It also may need to be increased, or can sometimes be shortened, according to the relative speed groups of the aircraft involved (allowing for the convergence or divergence of aircraft at take-off or shortly afterwards). It is possible to determine the minimum separation that is required between any two aircraft given the weight classes, speed

groups, departure routes and allocated runway(s), and the take-off time of the first of the aircraft (so that time-dependent separations can be considered). Any practical solution approach for the take-off problem must not assume that these separations are constants, even for a specific pair of aircraft.

There are distinct similarities between arrival sequencing problems [1], [2], [3] and take-off sequencing problems. However, it was shown in [4], [5], [6] that the control problem at Heathrow cannot be ignored. This situation is not unique to Heathrow [7]. The required separations, and hence the runway throughput, are very sensitive to the selection of aircraft which are available for take-off at that time, and especially to the routes upon which aircraft are due to depart. The fact that the route separations and take-off time-slots have major effects upon the delay for aircraft, [5], [6], means that approaches which categorise aircraft by weight category, such as [8], [9], [10] are not likely to be successful at Heathrow, even if the control problem could be ignored.

Controllers have to consider at least four objectives when considering which take-off sequences to enact. Firstly, runway throughput, or mean take-off delay, has to be considered. Secondly, the sequence cannot be prohibitively difficult to enact, in terms of the overtaking that is required and the time and workload involved in giving the necessary instructions to pilots. This is critical for the sake of ensuring safety. Thirdly, take-off time-slots apply to some aircraft and must be met. Finally, controllers must ensure that any single aircraft is not greatly penalised in the sequencing. This is important since characteristics of some aircraft (for example unusually small or slow aircraft) will necessarily require larger separations either before or after them. The re-sequencing must ensure that these aircraft are not just moved to the end of the take-off queue, where the larger separation would affect the take-off times of fewer aircraft.

Delay is used here as a measure of performance rather than throughput, since it is not only related to cost (for airlines and passengers, in terms of time as well as money) but also measures improvements that throughput measures can hide. The experiments performed for this paper consider half-day periods. The overall take-off throughput of the runway over a long period of time cannot exceed the rate at which departures are released from the stand. Even when busy, there can be times when the runway is starved of the correct types of aircraft which would be required to attain a good runway throughput. For example, if all waiting aircraft are northbound then no amount of sequencing ability will be able to reduce the separations between aircraft below two minutes. Throughput measures tend to consider only the take-off time of the last aircraft to take-off, rather than considering the take-off times of all aircraft, whereas delay-based measures take consideration to the sequencing of all aircraft. The advantages of measuring delay rather than throughput were discussed further in [6].

Due to congestion (or adverse weather conditions) in busy airspace or at busy destination airports, take-off time-slots exist for some aircraft. These are formulated as a Calculated Time Of Take-off (CTOT). An aircraft is not permitted to take off more than five minutes before this time and should not take off more than ten minutes after it. A limited number of further five minute extensions are available to controllers in order to reduce the number of renegotiations of CTOTs that are required. Thus, these CTOTs normally generate fifteen minute time-slots within which aircraft must take off, but on occasions can be stretched to twenty minutes, where necessary, by utilising an extension. Since extensions are limited and it is usually unknown how many extensions will be required later in the day, it is important for any sequencing to use as few as possible.

In order to predict when an aircraft would take off for a given take-off sequence, it is necessary to determine the earliest time at which it can do so. Predicting this prior to the aircraft reaching the holding area requires knowing the time at which an aircraft will be ready to push back from the stand and the taxi time from the stand to the runway. Due to the difficulty of predicting the time at which an aircraft will be ready for push back and the time required to reach the holding area, the current mode of operations is to release aircraft from the stands as soon as the workload for the ground controller will permit, get them to the runway holding area, and allow the runway controller to perform the take-off sequencing once they get

there. However, only a small minority of the runway controller's time can actually be spent considering the take-off sequence; the majority being spent communicating with pilots. Fast (mental) heuristic methods have to be used, therefore. These heuristics usually work extremely well, but sometimes throughput improvements can be obtained from considering more aircraft. In particular, it is sometimes possible to avoid future larger separations for aircraft which are still taxiing by knowing about them in advance and appropriately sequencing the aircraft which are already in the holding area.

3 A model for the sequencing problem

Let d_j denote the (actual or predicted) take-off time for aircraft j . Let et_j denote the earliest time at which aircraft j can take off given the earliest time it is reasonable to expect it to be able to reach the runway. The derivation of this value differs between the two problems which are considered in this paper and is discussed below. Let ec_j denote the earliest time at which aircraft j can take off within its CTOT time-slot, or a low enough value to not restrict the take-off time if j has no CTOT. For any ordered pair of aircraft i and j , where i takes off before j , let RS_{ij} denote the minimum required take-off time separation.

Let ts_j denote the position of aircraft j in the planned take-off sequence and let as_j denote the position of aircraft j in the First-Come-First-Served (FCFS) take-off sequence. The FCFS sequence can be considered to be that which would be generated if aircraft took off in the order in which they arrived at the holding area, there was no contention at the cul-de-sac or on the taxiways, and no stand holds were applied. This can be generated by considering the earliest pushback times for each aircraft then adding on the pushback durations and the (predicted or real) taxi durations (from pushback completion to reaching the holding area).

Measuring delay requires a base-line from which to measure. The base-line time, bt_j for aircraft j , is here defined as the time at which aircraft j historically reached the holding area, and the delay is defined as the duration from holding area arrival to take-off, $d_j - bt_j$.

The sequencing problem involves finding a good take-off sequence such that the constraints are met and the value of the objective function is minimised. There are three main constraints upon the take-off time for aircraft j . Firstly, j cannot take off before it can reach the runway, as shown by (1). Secondly, it cannot take off before the start of any CTOT time-slot, as shown by (2). Thirdly, all runway separation rules must be obeyed, as shown by (3).

$$d_j \geq et_j \quad (1)$$

$$d_j \geq ec_j \quad (2)$$

$$d_j \geq d_i + RS_{ij} \quad \forall i \text{ s.t. } ts_i < ts_j \quad (3)$$

The objectives for the sequencing element of the problem are threefold: reduce the delay for aircraft (or, equivalently, keep the runway throughput high), control inequity in the sequencing (i.e. do not allow any aircraft to be unduly penalised) and ensure that CTOT take-off slots are met where possible (and that extensions are met otherwise). Of course, maintaining safety is imperative through all of this. The separation rules ensure safety at take-off and the solution method for the control problem ensures that the achievement of any predicted sequence will not involve too much work since a controller's time is a limited resource so a controller would reject any sequence which took too long to achieve.

Let $C(j, d_j)$ denote a function to determine the cost associated with any failure to comply with CTOT time-slots if aircraft j takes off at time d_j . Function $C(j, d_j)$ will return 0 if d_j is within the CTOT for aircraft j , a penalty cost if d_j is not within the CTOT for d_j but is within an extension, and a much greater cost if d_j is too late to be within even a CTOT extension (so that any schedule for which this is the case will not be adopted).

Let $E(ts_j, as_j)$ denote a function to penalise positional inequity in the take-off sequence, for example penalising the positional delay (defined as $\max(ts_j - as_j, 0)$) or both positional

advancement and delay (for example, $\max(ts_j - as_j, as_j - ts_j)$). The effects of both of these factors were studied in [6], where factors related to the square of the positional delay and of the positional deviation were included in the objective function.

The objectives of the take-off sequencing can then be formalised by (4), where W_1 , W_2 and W_3 are weights for the three components, set so that CTOT compliance is of primary importance and delay is of secondary importance. The weight for equity is chosen so that small deviations are of little importance but large deviations are greatly penalised. For the purposes of this paper, the overall delay and CTOT compliance results are more important than the objective function which was used by the sequencing algorithm to obtain the results. Details of functions $C(j, d_j)$ and $E(ts_j, as_j)$ can be found in [6], [11] and [12] along with values for the weights which were used in each case.

$$W_1 C(j, d_j) + W_2 (d_j - bt_j) + W_3 E(ts_j, as_j) \quad (4)$$

A linear penalty for delay is shown in the objective function (4). Although this form was used for the runway sequencing decision support system, a non-linear penalty is currently being used for the TSAT allocation system, in addition to the factor $E(ts_j, as_j)$, to promote equity in the sequencing. When re-sequencing within the holding areas, experiments showed that the constraints imposed by the holding area structures favoured equitable schedules, since holding an aircraft for too long can block a part of the holding area, thus limiting the improvements that can be made to the sequencing of other aircraft. When take-off sequencing is performed by the TSAT allocation the system (in the absence of the equity-aiding constraints from the holding area structure), the delay is raised to the power 1.5 in the objective function, to further encourage the spreading of the delay more equitably between aircraft. Results are presented in this paper for the TSAT system with both linear and non-linear objective functions.

4 Take-off time prediction, for both systems

The same take-off time prediction method is used by both of the systems described in this paper. Aircraft in a partial or full take-off sequence are considered one at a time, in take-off order and the earliest time which will satisfy constraints (1), (2) and (3) is assigned to each aircraft. The order of evaluation is important since RS_{ij} can depend upon the time of day so, before d_j can be determined, it is necessary to know d_i for each aircraft i which takes off earlier than j . This method for take-off time prediction is known to be pessimistic (as discussed in [6] and [11]) in that it may predict take-off times which are later than would actually happen since the controllers are permitted to reduce some non-safety-related separations at their discretion.

5 Sequencing within the holding area

The physical layout of the holding area structure can be a major problem for re-sequencing aircraft once they reach the holding area. In some airports the physical layout can be restrictive enough to simplify the take-off sequencing problem [7], however the Heathrow holding areas are flexible enough to allow much re-sequencing, but not so flexible that all desirable re-sequencing is achievable (in which case they be ignored by the sequencing algorithm). It was identified in [5] that the removal of the constraints imposed by the holding area could make better take-off sequences achievable, and thus improve the overall delay at take-off.

The runway controller has to consider the locations at which the aircraft are currently situated, the instructions which they have been given so far, and the desirability of the various take-off sequences, then use this information to select a take-off sequence to enact. The holding area movement problem was thoroughly described in [6] and the system that was used in [4], [6], [11], [12] has been utilised for the experiments in this paper which use the runway controller decision support system.

To understand the effects of the holding area, it is useful to consider the way in which a directed graph model of the holding area can be used to determine the feasibility of re-sequencing. One such graph, for the 27R holding area which was used in these experiments, is presented in Fig. 1. The 27R holding area can be considered to consist of three parallel queues, with interchange points between them. Determining the feasibility involves positioning aircraft which are within the holding area at the node to which they are closest. Aircraft which are still on the taxiways are added to queues for the appropriate entrance nodes. The re-sequencing is deemed feasible if aircraft can enter the runway node(s) in the take-off order having moved one arc at a time without any node being occupied by more than one aircraft at any time.

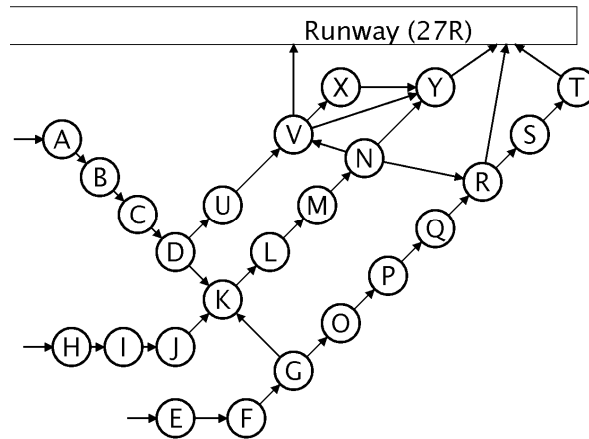


Figure 1. Directed graph of the 27R Holding area, taken from [6]

The sequencing problem faced by the runway controller is complicated by a number of factors. Firstly, the value of e_{t_j} can change depending upon the way in which the control problem is handled. For example, allocating a longer path through the holding area to aircraft j would be expected to increase the value of e_{t_j} , since aircraft will take longer to reach the runway. The value of e_{t_j} can be considered to be a composition of the holding area arrival time and the length of time necessary to traverse the holding area and line up for the runway and can, therefore, only be determined for aircraft once the control problem has been considered. However, the paths which should be allocated to aircraft will depend upon the target take-off sequence, since short paths need to be allocated to those aircraft which will have less time available to traverse the holding area. Furthermore, it is important for sequencing to be achievable in a manner which will be sensible to the runway controller and keep the workload low.

6 Solution method for sequencing at the holding area

The solution method for this problem was described in [4], [6], [11] and [12]. A tabu search meta-heuristic is used to investigate the potential take-off sequences, utilising moves to swap the positions of aircraft, shift multiple aircraft forwards or backwards in the sequence, or randomly re-sequence a continuous sub-sequence of two to five aircraft. Each sequence is evaluated individually. Paths through the holding area are heuristically assigned to aircraft according to the overtaking which they need to perform. This is performed in such a manner that e_{t_j} will usually be as low as possible for any aircraft j for which e_{t_j} is the binding constraint upon the take-off time. Partial sequences of movement are generated for the nodes close to the entrances and exits of the holding area, and counts of aircraft and spare nodes are maintained. A look-ahead algorithm is then used to move aircraft within the holding area graph, taking

consideration of the aircraft counts in order to avoid blocking. The sequence is rejected if either it is infeasible or the controller is unlikely to accept the holding area movement which would be required to achieve it (for example, if the required paths would not appear sensible, or the sequencing of movement is too complex). This ensures that any adopted take-off sequence will be relatively easy to achieve.

The outputs of the control problem stage consist of an allocated path for each aircraft, an earliest take-off time and full details of how the re-sequencing would be achieved. The derivation of the earliest take-off time (et_j for aircraft j) takes into consideration the position of each aircraft, the allocated paths, the arrival times of aircraft at the holding area and any delay which is necessary due to waiting for other aircraft to move out of the way. The movement details are used by the simulation which evaluates the performance of the system in order to predict the positions of aircraft over time. They are also very useful for watching playback of the movement, to ensure that the system is working as intended and that all movement within the holding area is both as expected and acceptable. Once a take-off sequence has been determined to be feasible, and the et_j values for each aircraft have been determined, the sequence is evaluated to determine the take-off times (where d_j is assumed to be the earliest value which will satisfy (1), (2) and (3)) and the consequent cost of the sequence, as measured by (4).

7 Considering the sequencing earlier in the departure system

Given the difficulties of re-sequencing at the holding area, it is worth considering whether the re-sequencing can be performed any earlier in the departure system. To determine a take-off sequence and take-off times, it is necessary to know the earliest times at which each aircraft can reach the runway (et_j for each aircraft j). This requires a prediction for each aircraft for both the taxi time from the stand to the runway, and an earliest pushback time from the stand. It has not previously been feasible to determine earliest pushback times, but recent innovations at Heathrow will make it more so. A Collaborative Decision Making (CDM) project is currently being implemented at Heathrow. For the first time, this will automate the sharing of information between airlines, ground handlers and controllers, potentially enabling all parties involved to make better decisions. For the purposes of this research, this means that earliest pushback times (called TOBTs, Target Off-Block Times, the times at which the airline believes the aircraft will be ready for pushback) will be provided by airlines to controllers. In combination with taxi durations (discussed below), this will make earliest take-off times more predictable. Controllers can then use the predicted take-off times to generate TSATs (Target Start-At Times) to pass back to airlines specifying the time at which the aircraft will actually be permitted to push back. The CDM project means that airlines gain greater visibility of expected delays, and controllers gain the knowledge that is necessary to predict take-off delays and assign appropriate stand holds.

7.1 Taxi-out duration

Taxi-out duration (or time) often refers to the time from leaving the stand to line-up on the runway, and may be used as a measure of the delay that aircraft should expect from stand/gate to take-off [13]. Since the queueing time can form the major element of the delay for aircraft, this measure of taxi time can be highly dependent upon the state of the airport at the time. This definition is, thus, not necessarily the most useful measure of the taxi time if the aim is to determine the earliest time at which a take-off could be scheduled.

At Heathrow, the taxi duration can be considered to have three main components: a delay from wishing to pushback to actually having completed pushback (consisting of a pushback duration plus any waiting time due to contention with other aircraft, as discussed later), the travel time from the pushed back position to the holding area queue, and some queueing and lining up time at the runway. The queueing time at the runway will depend upon the take-off sequence and is thus relatively predictable for a known take-off sequence, as long as the arrival

time at the holding area is known and sufficient time is allowed for aircraft to traverse the holding area and line up for take-off. The taxi time from cul-de-sac to runway is believed to be relatively predictable, once the delays around the stands and at the runway are removed from the problem. Thus, there is a need to predict the final component of the taxi duration, the delay around the stand.

7.2 Cul-de-sac contention

Most stands at Heathrow are grouped around the terminal piers, as illustrated in Fig. 2. Cul-de-sacs are formed between the piers, with one end of the cul-de-sac opening out onto the taxiways and the other very close to the main part of the terminal. Due to these cul-de-sac structures, aircraft may block other aircraft from pushing back or from leaving the cul-de-sac until they have started their engines and moved out of the way. This causes delays for the blocked aircraft, which can traditionally appear as unpredictable increases in the taxi times. By explicitly modelling the contention within the cul-de-sac, the unpredictability of these cul-de-sac delays can be greatly reduced.

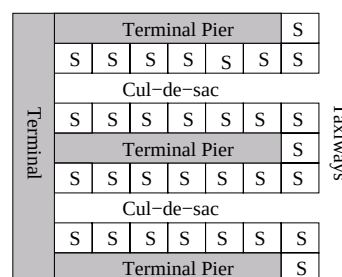


Figure 2. Stylised diagram of how cul-de-sacs are formed between the stands (labelled S) at the terminal piers

8 TSAT allocation

The TSAT allocation problem involves assigning pushback times (Target Start-At Times, TSATs) to aircraft such that excess delay can be absorbed at the stand without reducing the runway throughput. The solution method that has been implemented for this problem involves predicting a take-off sequence for aircraft while they are still at the stand, then using this to determine how much stand hold to assign to each aircraft. The main elements of such a system are a method to determine the amount of cul-de-sac delay that aircraft will expect (by modelling the contention between aircraft which are in the same cul-de-sac or at stands which are close together), a consideration of the take-off sequencing problem (to determine a good take-off sequence), and a method to determine the appropriate stand hold to allocate to an aircraft given a predicted take-off time (so that excess delay is absorbed at the stand but enough slack is available to allow for variation between real and actual taxi times).

The aim is to absorb any delay beyond a specified threshold value (labelled IRH_j for aircraft j in the model presented later) at the stand instead of the runway, rather than to attempt to absorb all of the delay at the stand. The characteristics of good take-off sequences mean that they tend to group aircraft by weight class, and that aircraft alternate departure routes. This means that aircraft in alternate positions in the take-off sequence often have similar (but not usually exactly the same) characteristics. Alternative good (but not necessarily quite as good) take-off sequences can often be generated from a good take-off sequence by switching the positions of aircraft which take off at similar times, especially those which take off two positions apart in the take-off sequence.

The TSAT system can only have an adverse effect upon the potential take-off sequencing if aircraft are held at the stands to the point where the runway controller is starved of the

appropriate aircraft to achieve a good take-off sequence. At quieter times, when delay at the runway is low, stand holds will very rarely get allocated, in which case TSAT allocation will have no adverse effect upon the potential sequences. As the departure system get busier and delay increases, the benefits of the system become more apparent as stand holds increase. However, the threshold value for the runway hold ensures that alternative aircraft are usually available when unexpectedly long taxi times mean that a specific aircraft does not arrive in time. Thus, good alternative take-off sequences should be available when the predicted take-off sequence cannot be achieved. The potential for the achieved take-off sequence to deviate from the predicted take-off sequence means that the TSAT system needs to find good take-off sequences, but there is not necessarily a benefit from finding the optimal take-off sequence while aircraft are still at the stands.

The primary benefit of the TSAT allocation system is to reduce the waiting time with engines running while ensuring that the better take-off sequences are no less likely to be achieved than without the stand holds. However, a potential secondary benefit is to simplify the problem for the runway controller. The reduction in aircraft at the holding area at busy times should reduce the complexity of the sequencing problem, reducing the amount of blocking within the holding area and enabling the controllers to achieve the better take-off sequences which may not normally be available due to the restrictions of the physical layouts of the holding areas.

9 Take-off sequencing with cul-de-sac contention

The take-off sequencing aspect of the problem is very similar to that considered by the runway controller. The same objectives (as expressed by (4)) and constraints (expressed by (1), (2) and (3)) apply to the sequencing element of the problem. The main differences in this case are, firstly, that the control problem at the holding area is ignored (on the basis that so few aircraft will actually be at the holding area that any re-sequencing there will be achievable) and, secondly, that the earliest take-off time will depend upon the taxi duration and the time at which the aircraft leaves the cul-de-sac, which will itself depend upon the contention within the cul-de-sacs. With the earlier definition for bt_j as the historic holding area arrival time, $d_j - bt_j$ will be the sum of the stand delay and the time spent in the holding area. In a real system, in the absence of historic holding area arrival times, the equation $bt_j = ept_j + pd_j + td_j$ would be used to predict a holding area arrival time with no stand hold, with the following definitions for ept_j , pd_j and td_j .

Let pt_j denote the known or predicted pushback time of aircraft j . Let pd_j denote the known or predicted pushback duration for aircraft j . This is the time taken to push back, start engines, and be ready to taxi towards the runway. Let td_j denote the taxi duration for aircraft j , from the cul-de-sac to the holding area. Let ept_j denote the earliest pushback time for aircraft j (the TOBT). Let ct_i denote the cul-de-sac time for aircraft i , defined as the time at which i will be able to commence its taxi towards the runway without being blocked from doing so by other aircraft. Similarly, let ct_j denote the cul-de-sac time for aircraft j . Let cs_i and cs_j denote the positions of aircraft i and j (respectively) in the sequence in which aircraft commence their taxi towards the holding area, so that aircraft with earlier cul-de-sac times have lower values for cs_i .

$$et_j = ct_j + td_j + MRH_j \quad (5)$$

$$ct_j \geq ept_j + pd_j \quad (6)$$

$$ct_j \geq ct_i + MS_{ij} \quad \forall i \text{ s.t. } cs_i < cs_j \quad (7)$$

$$pt_j = ct_j - pd_j \quad (8)$$

Inequalities (1), (2) and (3) still apply to the take-off times. The value of et_j is now related to the cul-de-sac time of aircraft j , as shown by (5), where MRH_j is a minimum runway hold

that can be allocated to aircraft j . The value of MRH_j should be large enough to allow for the time to traverse the holding area and line up (this normally requires less than a minute) and may also include some slack to allow for taxi time variability. Given that this value is used to determine how early an aircraft can be scheduled to take off, it is important not to set it to too great a value. Doing so would involve either an unnecessary wait for take-off if the sequence was adopted or, more likely, the controller would adopt a (better) sequence which would avoid the unnecessary delay.

The contention within the cul-de-sac can be modelled as minimum separation requirements between the cul-de-sac times for aircraft, so that each aircraft can be moved out of the way before another aircraft has to use that section of a cul-de-sac, as described in [14]. Let MS_{ij} denote a minimum cul-de-sac time separation that is required between aircraft i and j , when i leaves the cul-de-sac prior to j . The constraints upon cul-de-sac times can then be described by (7). The cul-de-sac times are related to the earliest pushback times (or TOBTs) by (6). Finally, given a cul-de-sac time, it is sensible for an aircraft to push back just in time to meet its calculated cul-de-sac time, so (8) can be used to determine a pushback time from a cul-de-sac time. The output of the take-off sequencing stage is, thus, a take-off time and feasible pushback time for each aircraft.

10 The push-back time allocation sub-problem

The pushback time allocation problem can be considered as one of assigning the optimal cul-de-sac times given the predicted take-off times. Rather than assign stand holds to absorb all of the delay, in which case achieving the target take-off schedule would depend upon all aircraft achieving the predicted pushback times and taxi times, the aim is to absorb all delay beyond a specific threshold value. An ideal runway hold, IRH_j for aircraft j , is determined, which specifies the target time prior to take-off at which j should be planned to arrive at the holding area - providing that sufficient time is available. In other words, for any aircraft j , IRH_j is the threshold duration beyond which hold should be absorbed as stand hold, and below which it should be absorbed as runway hold. Obviously, the values that are used for IRH_j will affect the balance between the expected runway and stand hold.

$$ict_j = \max(ept_j + pd_j, d_j - IRH_j - td_j) \quad (9)$$

$$\sum_j (100 \max(0, ct_j - ict_j)^{1.1} + \max(0, ict_j - ct_j)^{1.1}) \quad (10)$$

Given an ideal runway hold, an ideal cul-de-sac time, ict_j , can be determined for each aircraft j using (9). A cost can then be associated with any deviation of the cul-de-sac time for aircraft j from the ideal time, and the objective for the pushback time allocation stage is to reduce the sum of these costs for all aircraft under consideration (10). The nonlinear cost of deviation is again used to slightly favour more equitable distributions of any necessary deviations. Weights of 100 and 1 are used for the two elements in (10) so that the algorithm favours aircraft pushing back early (increasing the runway hold time) rather than late.

10.1 Ideal and minimum runway hold values

The values that are used for MRH_j and IRH_j should be determined according to the reliability of the taxi time data. IRH_j and MRH_j have different but complementary effects. MRH_j can be used to ensure slack for the taxi times, while IRH_j can be used to favour schedules with more slack where possible, without prohibiting good schedules which do not allow so much slack. For the purpose of this paper, a minimum hold of two minutes was chosen so that the earliest take-off times are compatible with those for the runway sequencing decision support system values. An ideal runway hold of five minutes was chosen for all aircraft. This is short enough

that some stand hold will be allocated to aircraft but large enough to provide a pool of a few aircraft at the holding area at most times.

11 Solution method for the TSAT problem

The implemented solution method for the TSAT allocation problem first generates a take-off sequence and good feasible take-off times, then uses this to assign appropriate stand holds to aircraft. The two sub-problems are solved separately, although (1) and (5) mean that cul-de-sac times have to be considered when considering the take-off sequencing, so that the second stage has at least one feasible solution. The cul-de-sac times depend, of course, upon the cul-de-sac sequence (due to (7)). The fact that cul-de-sac separations are both asymmetric and do not obey the triangle inequality means that the best cul-de-sac sequence does not necessarily match the planned take-off sequence, however, only the feasible rather than optimal cul-de-sac times are required at the take-off sequencing stage.

The developed solution method first heuristically generates an initial take-off sequence (based upon allocated CTOTs and TOBTs), then incrementally improves this using a branch-and-bound algorithm. The branch-and-bound algorithm is applied within a rolling window through the sequence (from earliest take-off to latest take-off), finding the optimal sequence of the aircraft within the current window. It assumes that the aircraft prior to the window have a fixed take-off sequence (and take-off times) and ignores aircraft beyond the end of the window. This solution approach was described in more detail in [14].

The branch and bound algorithm that is applied each window considers partial take-off sequences, starting from the first take-off within the window and consisting of consecutive take-offs. Aircraft are added to the sequence one at a time. As aircraft are added, a take-off time is determined for the aircraft, along with a feasible cul-de-sac time. If the cul-de-sac time is the limiting factor upon the take-off time, the cul-de-sac sequencing problem is solved to determine the cul-de-sac times which will result in the earliest take-off time for the current aircraft. In this way, the take-off times for aircraft in partial take-off sequences can be determined, along with a consequent cost for the partial sequence. A lower bound is then determined for the cost of the remaining aircraft in the window which have not yet been added. If the resulting lower bound for the cost of any sequence with this partial take-off sequence as a prefix exceeds the cost of a previously evaluated full take-off sequence, the partial sequence is pruned from further consideration.

The fact that exchanging two aircraft in a take-off sequence can often generate a very similarly valued take-off sequence is a useful property in that it allows recovery from unpredictable taxi times, but it is a problem for the branch and bound algorithm. There will usually be a large variety of similarly costing schedules with different initial partial take-off sequences, thus the potential for early pruning of partial schedules is limited. However, take-off times can only be predicted accurately when a partial sequence consists of a consecutive set of aircraft with some known starting time, so it is necessary to build-up sequences in this way. Although the branch and bound significantly reduces the number of sequences which must be considered, the solution time still rises rapidly as the window size increases.

In the presence of CTOTs it may be necessary for specific aircraft to move a long way in the take-off sequence (as shown in [6]) and, although the heuristic pre-sequencing helps with this, it is not always sufficient. In order to allow aircraft to be advanced a large number of places when necessary, the window size is kept relatively low and the rolling window is applied multiple times to provide further incremental improvements.

Once take-off times and feasible cul-de-sac times have been determined, a second stage algorithm is applied to optimise the cul-de-sac times, as defined by (10). Each aircraft will have an ideal cul-de-sac time (calculated from its take-off time, using (9)), an earliest cul-de-sac time (calculated from its TOBT, using (6)) and a latest cul-de-sac time (provided as input information or calculated from the take-off time as the latest time which would allow (5) to hold). Despite the non-linear costs of deviation from ideal times, this is usually a relatively simple problem to solve, since it can be decomposed into small sub-problems of mutually

dependent aircraft. Since TSATs must lie on minute boundaries, the number of possible TSATs is usually very limited. A simple branch and bound algorithm is again used.

12 Comparing solution methods

The solution methods for the two take-off sequencing problems differ. Sequencing at the runway is performed using a meta-heuristic search. Sequencing at the stands is performed using a branch and bound method within a rolling window. This is due to fundamental differences in the problems. Firstly, sequencing at the runway has to be extremely fast, so a heuristic method is required, whereas the longer time available for TSAT allocation (and the ability to ignore the structure of the holding area when sequencing at the stands) makes the branch-and-bound possible within a window. In a dynamic situation, the TSAT allocation problem will have to cope with latest TSAT constraints as well as earliest TSAT constraints. These windows can make it very difficult to find feasible solutions that meet the constraints, limiting the effectiveness of a meta-heuristic search. Although the branch-and-bound solution method works within a window, in the presence of latest TSAT constraints it has to also take into consideration the aircraft outside of the current window in order to ensure feasibility of later TSAT allocations. These considerations are irrelevant for the experiments described in this paper, but important motivations for the system design when considering the real problem.

The second reason for the difference is the value of a partial take-off sequence. Issues related to the control problem at the holding area mean that partial take-off sequences are of limited value for the decision support system for the runway controller. Adding new aircraft to the sequence may require path changes for previous take-offs, which may alter take-off times. Furthermore, the movement of previous take-offs may greatly restrict the possibilities for re-sequencing later take-offs which had to be overtaken. In contrast, when sequencing at the stands, the value of a partial take-off sequence is a good indication of the value of a full take-off sequence, thus evaluating partial sequences within the branch-and-bound algorithm can avoid a lot of re-calculation of cul-de-sac times.

13 Comparative results

Experiments were performed using two datasets which had sufficient information to be used by both the TSAT allocation system and the decision support system for the runway controller. All experimental data was for the 27R holding area and avoided runway change-over times. The first dataset contained details of 345 consecutive take-offs and the second 341 consecutive take-offs.

The decision support system for the runway controller was executed within the simulation of the departure system that was described in [4], [6], [11]. Each evaluation involved a number of iterations of the simulation. In each iteration, the system was presented with a problem to solve before the time was advanced by one minute, and the new, consequent, problem was presented in the next iteration. Once all aircraft in the dataset had taken off, the resulting overall take-off sequence was evaluated. The decision support system has a stochastic element, so each evaluation was executed ten times and the mean results are presented. It should be noted that the results for each evaluation were very similar.

For comparison, the TSAT allocation system was applied to the static problem, considering the entire dataset of over 340 aircraft at once. Rolling windows of various window sizes were applied and the resulting take-off sequences and TSATs were evaluated. Of course, there is no intention to communicate to the controllers the predicted take-off times or sequences, but consideration of the internally predicted sequences is useful for comparing the sequencing at the stands against that at the runway.

For the runway controller decision support system, historic holding area arrival times were used and the system was permitted to schedule aircraft to take-off from two minutes after the historic holding area arrival time. For the TSAT allocation system, historic pushback times were used as earliest pushback times. Historical taxi times were also used, so that an aircraft

without any stand hold, and that pushed back as soon as it could, would have the historic holding area arrival time. A two-minute minimum runway hold was applied, so that an aircraft could not take off within two minutes of its predicted holding area arrival time. Thus, with no stand hold, the two systems had the same earliest take-off times for aircraft, although stand contention had to be considered for the TSAT allocation system.

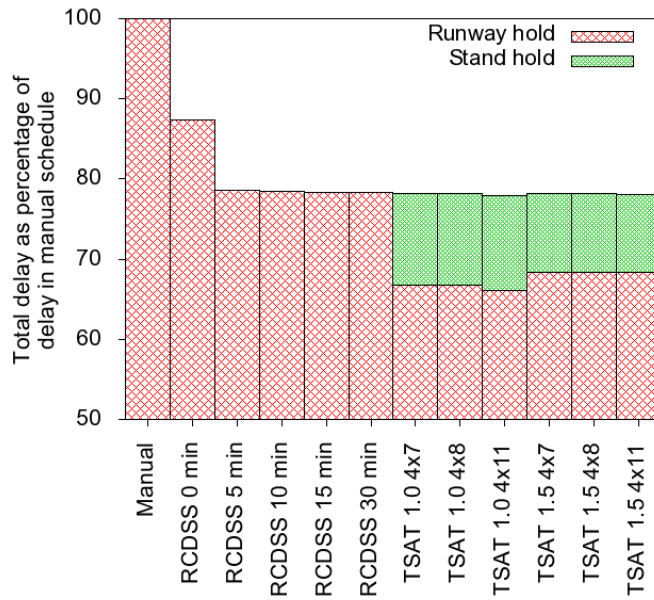


Figure 3. Comparative delays, dataset 1

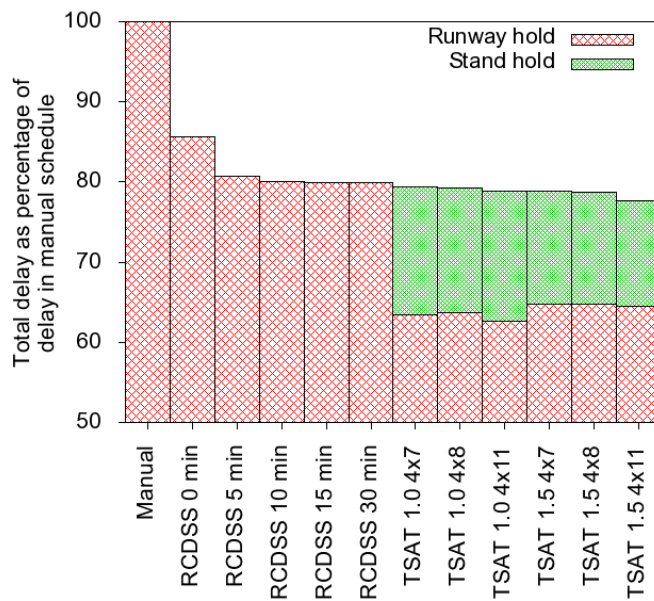


Figure 4. Comparative delays, dataset 2

There were three aims for these experiments. Firstly to obtain an idea of how much potential benefit could be gained from a decision support system which was able to act upon more information than a controller could be expected to cope with. Secondly, to identify how the take-off sequencing elements of the two systems performed when compared against each other. Thirdly, to obtain insight into how much of any prospective delay could reasonably be absorbed at the stands prior to starting the engines.

Figs. 3 and 4 illustrate the performance of the decision support system for the runway controller for each of two datasets. All values are shown as percentages of the delay obtained by the controllers for the manually produced take-off sequences with real take-off times. For reference, the manual results had mean delays per aircraft of 311s and 340s for datasets 1 and 2 respectively. In each case, the pessimistic take-off time prediction system was used to generate the take-off times for the automatically generated take-off sequences. i.e. the earliest take-off time which will meet constraints (1), (2) and (3) is assumed for each aircraft.

The results labelled 'Manual' show the real delay in the manually produced sequence. The 'RCDSS 0 min', 'RCDSS 5 min', 'RCDSS 10 min', 'RCDSS 15 min' and 'RCDSS 30 min' results show the performance of the decision support system for the runway controller with knowledge of aircraft 0, 5, 10, 15 and 30 minutes respectively prior to the aircraft arriving at the holding area. The 'TSAT 1.0 4x7', 'TSAT 1.0 4x8' and 'TSAT 1.0 4x11' results show the performance of the TSAT allocation system with linear costs for the delay, 4 passes of the rolling window and window sizes of 7, 8 and 11 aircraft respectively. The 'TSAT 1.5 4x7', 'TSAT 1.5 4x8' and 'TSAT 1.5 4x11' are identical to the 'TSAT 1.0 4x7', 'TSAT 1.0 4x8' and 'TSAT 1.0 4x11' experiments respectively except that a non-linear cost was applied for delay (the number of seconds delay in objective function (4) was raised to the power of 1.5).

The performance of the decision support system for the runway controller depends upon how early the aircraft enter the system and how many aircraft are considered. This was discussed further in [6] and [15]. In particular, the tested configuration of the system assigned paths to aircraft as soon as they arrived at the holding area. This meant that the system performance when it was only made aware of aircraft at the point at which they arrived at the holding area (i.e. 'RCDSS 0 min') was significantly worse than when the system had knowledge of them prior to holding area arrival. System performance was slightly better when more aircraft could be considered (i.e. when aircraft entered the system earlier) but the differences were relatively small. The simulations predicted that the decision support system was able to find schedules which reduced the delay by over 20%, a significant saving in time and fuel.

The performance of the take-off sequencing element of the TSAT allocation system was slightly better than that of the decision support system for the runway controller, even with low window sizes. This was expected for two reasons. Firstly, consideration of the entire static problem provides the TSAT allocation system with greater knowledge of aircraft than is available to the runway controller's decision support system. Secondly, under this formulation, it is possible to hold aircraft at the stand for arbitrarily long if necessary without reducing the sequencing flexibility, whereas the decision support system for the runway controller has to work within the holding area constraints and it was previously shown in [5] that the holding area constraints affect the performance of the sequencing.

Since stand delay does not contribute to fuel usage, the results for the TSAT system are significantly more environmentally friendly than those for the runway sequencing. Not only are similar total delay reductions achieved, but a significant amount of the delay can be achieved as stand hold rather than delay at the holding area. For the experiments performed for this paper, this resulted in an additional 9.9% to 16.1% of the full delay of the manual schedule being absorbed at the stand.

Interestingly, considering equity within the objective function by using a non-linear cost for delay actually slightly reduces the overall delay in the schedule for dataset 2, rather than increasing it. More importantly, the amount of delay which can be absorbed at the stand is higher for the linear delay cost. The mean delay values for datasets 1 and 2 were 4.1 and 4.5 minutes respectively. Only aircraft with greater than a 5 minute runway hold will receive any

stand hold. A less equitable take-off sequence will have more deviation of delays away from the mean, so more aircraft with a delay greater than five minutes, thus more aircraft will have stand holds. However, these schedules will also have more aircraft with lower runway holds so would be expected to be less robust due to the lower slack and smaller pool of aircraft at the holding area. This robustness effect is another reason for promoting equity of delay and would comprise as interesting area for further research. Tuning of these parameters for the specific situation at Heathrow is a planned future stage of the TSAT implementation project.

It should be noted that these delay improvements in the automated schedules were not at the expense of CTOT compliance. The manually produced schedules missed 5 and 6 CTOTS respectively for datasets 1 and 2. In all cases, the automated schedules missed 1 CTOT for dataset 1, and between 3 and 5 for dataset 2. 'RCDSS 0 min' missed 5 CTOTs for dataset 2. 'RCDSS 5 min' and 'RCDSS 10 min' missed 4 CTOTs for dataset 2. 'RCDSS 15 min', 'RCDSS 30 min' and all of the TSAT results missed 3 CTOTs for dataset 2. Thus, in all cases, the automated schedules required fewer CTOT extensions than the manual schedules.

14 Potentially linking the two systems

Although the two models and systems could be used independently, better results could reasonably be expected from using both simultaneously. The primary benefit of the decision support system for the runway controller is that it provides visibility of the taxiing aircraft, and enables sequencing decisions to take into account more aircraft - with consequent reductions to the total delay. The presence of a TSAT allocation system should ensure that the holding area is not congested with aircraft which have long waits. Given that a good potential take-off sequence was originally planned by the TSAT system, there may be some benefit from giving visibility of this intended sequence to the runway controller. For example, a link between the two systems could be used to provide the decision support system for the runway controller with an initial sequence based upon that generated by the TSAT system, helping to guide the meta-heuristic search. The decision support system for the runway controller would then take into account the deviations between the predictions that the TSAT system made and what happened in reality and produce a recommended take-off sequence for the runway controller. The two systems would, therefore, potentially complement each other.

15 Conclusions

This paper presented a summary of some of the recent research involving the departure system that has taken place for London Heathrow airport. Two examples of take-off sequencing were considered. Each was applied at a different part of the departure process and, consequently, had different constraints upon it. In both cases the problem was complicated by not physically being able to achieve all desired take-off sequences due to contention between aircraft at the runway holding areas or in the cul-de-sac.

The first problem considered take-off sequencing from the point of view of the runway controller, and the second considered performing predictive sequencing earlier in the departure system, while aircraft are still at the stands, and (rather than doing anything with the sequence itself) using this to allocate an appropriate stand hold to aircraft so that an appropriate amount of any expected delay is absorbed prior to starting the engines.

The sequencing elements of the TSAT system and the decision support system for the runway controller were finding take-off sequences with very similar total delays, although the TSAT system performed slightly better. Of course, these results assumed predictable taxi times and pushback times. A pessimistic take-off time prediction system was used, and unnecessarily large minimum runway hold values were specified, so a real controller could potentially perform even better if given the appropriate information in a timely manner.

These results show that either of the two described systems could potentially provide great benefits at Heathrow, which is precisely why the TSAT system was developed. Moreover, the benefits of the TSAT system are more than just a potential for a reduction in the

total delay, since a considerable amount of the remaining delay could potentially be absorbed at the stand. The improved selection of aircraft to provide to the runway controller which should result from the implementation of a TSAT system should aid the runway controllers by reducing the congestion which can prevent better sequences from being enacted.

Acknowledgements The authors wish to thank NATS (formerly National Air Traffic Services Ltd) for partially funding this work and for providing time with their controllers and analysts to ensure that the work is relevant for the real situation at Heathrow. We would also like to thank the Engineering and Physical Sciences Research Council for providing the majority of the funding.

References

1. A. T. Ernst, M. Krishnamoorthy, and R. H. Storer, "Heuristic and exact algorithms for scheduling aircraft landings," *Networks*, vol. 34, pp. 229–241 (1999)
2. J. E. Beasley, M. Krishnamoorthy, Y. M. Sharaiha, and D. Abramson, "Scheduling aircraft landings -the static case," *Transportation Science*, vol. 34, pp. 180–197 (2000)
3. H. Pinol and J. Beasley, "Scatter search and bionomic algorithms for the aircraft landing problem," *European Journal of Operational Research*, vol. 171, pp. 439–462 (2006)
4. J. A. D. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson, "A meta-heuristic approach to aircraft departure scheduling at London Heathrow airport," in *Computer Aided Systems of Public Transport*, ser. Lecture Notes in Economics and Mathematical Systems, M. Hickman, P. Mirchandani, and S. Voss, Eds. Springer, Berlin (2008)
5. J. A. D. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson, "An examination of take-off scheduling constraints at London Heathrow airport," in *proceedings of the 10th International Conference on Scheduling of Public Transport (CASPT)*, Leeds, UK (2006)
6. J. A. D. Atkin, "On-line decision support for take-off runway scheduling at London Heathrow airport," Ph.D. dissertation, University of Nottingham (2008)
7. M. A. Bolender and G. L. Slater, "Analysis and optimization of departure sequences," in *proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit*, Denver, CO, August 2000, pp. 1672–1683
8. I. Anagnostakis and J.-P. Clarke, "Runway operations planning, a two-stage heuristic algorithm," in *proceedings of the AIAA Aircraft, Technology, Integration and Operations Forum*, Los Angeles, CA, October 2002, report AIAA-2002-5886
9. I. Anagnostakis and J.-P. Clarke, "Runway operations planning: a two-stage solution methodology," in *proceedings of the 36th Hawaii International Conference on System Sciences (HICSS-36)*, Hawaii, vol. 3, January 2003
10. H. N. Psaraftis, "A dynamic programming approach for sequencing groups of identical jobs," *Operations Research*, vol. 28, no. 6, pp. 1347–1359 (1980)
11. J. A. D. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson, "Hybrid meta-heuristics to aid runway scheduling at London Heathrow airport," *Transportation Science*, vol. 41, no. 1, pp. 90–106 (2007)
12. J. A. D. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson, "On-line decision support for take-off runway scheduling with uncertain taxi times at London Heathrow airport," *The Journal of Scheduling*, vol. 11, no. 5, pp. 323–346 (2008)
13. H. R. Idris, J.-P. Clarke, R. Bhuvra, and L. Kang, "Queuing model for taxi-out time estimation," *Air Traffic Control Quarterly (ATCA Publications)*, vol. 10, no. 1, pp. 1–22 (2002)
14. J. A. D. Atkin, E. K. Burke, and J. S. Greenwood, "The TSAT Allocation System at London Heathrow: The Relationship Between Slot Compliance, Throughput and Equity," in *proceedings of the 11th Conference on Advanced Systems for Public Transport (CASPT)*, Hong Kong, July 2009.
15. J. A. D. Atkin, E. K. Burke, J. S. Greenwood, and D. Reeson, "The effect of the planning horizon and the freezing time on take-off sequencing," in *proceedings of the 2nd International Conference on Research in Air Transportation (ICRAT)*, Belgrade, Serbia and Montenegro, June 2006.

‘Strong’-‘Weak’ Precedence in Scheduling: Extended Order Implications

Moshe Dror · George Steiner

Abstract We examine computational complexity implications for scheduling problems with job precedence relations with respect to strong precedence versus weak precedence. We propose a consistent definition of strong precedence for chains, trees, and series parallel graphs. Using modular decomposition for partially ordered sets (posets), we restate and extend past complexity results for chains and trees as summarized in [5]. Moreover, for series parallel graphs we establish new computational complexity results for strong precedence constraints for single and multiple machine problems.

1 Introduction

Job scheduling with precedence constraints has been at the forefront of scheduling research since its inception. Application of scheduling with precedence relations can be traced to PERT scheduling, early operating systems task scheduling with respect to its stability, and much more. In terms of solvability (e.g., finding an optimal schedule in polynomial time), adding a precedence relation to otherwise independent jobs in general results in a harder problem than the original one. Given a set of jobs J together with a partial order $<_P$ defined on $J \times J$, the common interpretation of precedence constraints (prior to Dror et al., [6,7]) was the following: for any $i, j \in J$, $i <_P j$ means that job i has to be completed before the processing of job j can start. This interpretation of the precedence constraints represented by partial order $<_P$ allows any amount of time delay between the completion of job i and the start of job j . A different interpretation of job precedence has been proposed in [6] and [7], recognizing that in some cases, further constraining the start time of job j with respect to the completion time of job i is justified in practice and implies a different problem and a different solution. Simply put, asking for a job j to *immediately* follow the completion of job i links the two jobs in a strong precedence. A weak precedence relation reflects

Moshe Dror

MIS, Eller College of Management, The University of Arizona, Tucson, AZ 85721

E-mail: mdror@eller.arizona.edu

George Steiner

DeGroote School of Business, McMaster University, Hamilton, Ontario, Canada L8S 4M4

E-mail: steiner@mcmaster.ca

the ‘old’ definition of precedence. For chained jobs, strong precedence implies that the jobs must be processed consecutively without the option of inserting a job not in the chain in-between the chained jobs. This is exactly the way the QMS algorithm schedules chained jobs in a manufacturing of mattresses as described by Assad et al. (1995) [1]. As a further motivation for our pursuit of the strong/weak precedence distinction, we note that some scheduling problems are solvable in polynomial time with strong precedence even though the weak precedence version is NP-hard and vice versa [6].

The strong/weak precedence distinction for chained jobs has been introduced in [6] and [7]. In [8] this distinction has been introduced for in-tree and out-tree precedence graphs. However, the strong chain precedence as defined in [6] and [7] was not consistent with the strong precedence tree relation as defined in [8]. This definitional inconsistency has been rectified in the present paper. In the past, series parallel precedence relations have not been examined from the prospective of strong versus weak job precedence. In this paper we define strong precedence for series parallel graphs consistent with strong precedence for chains and trees using modular decomposition of posets and develop new results for scheduling problems. For completeness, we also provide an abbreviated survey of the state-of-the-art for some scheduling problems reviewing their complexity with respect to strong and weak precedence relations. The outline of the paper is as follows: In Section 2 we state the extended strong precedence consistent for chains, trees, and series parallel precedence graphs. Section 3 examines single-machine problems. Section 4 studies multi-machine problems. In Section 5 we summarize our findings.

2 Extended Strong Precedence Constraints

Definition 1 Let $P = (J, <_P)$ be a partially ordered set (a poset). Let $\prec$ denote the immediate predecessor relation in P . That is, $\prec$ represents the transitive reduction of $<$. For $j \in J$ denote by $ImSuc(j) \triangleq \{k | j \prec k\}$ and by $ImPred(j) \triangleq \{k | k \prec j\}$. We say that $A \subseteq J$ is an *in-module* in P if $\forall i, j \in A$ we have $ImSuc(i) \cap (J \setminus A) = ImSuc(j) \cap (J \setminus A)$. A subset $B \subseteq J$ is an *out-module* in P if $\forall i, j \in B$ we have $ImPred(i) \cap (J \setminus B) = ImPred(j) \cap (J \setminus B)$. We say that $A \subseteq J$ is a *bi-module*, if it is both an in-module and an out-module.

Note that the empty set, any single point $i \in J$, or the whole set $A = J$ are trivial bi-modules. Furthermore, we note that if $A \subseteq J$ is an in-module in a general poset P , then $\forall i \in A$ we have either $ImSuc(i) \cap (J \setminus A) = \emptyset$ or $ImSuc(i) \cap (J \setminus A)$ is an antichain. Similarly, if $B \subseteq J$ is an out-module in a general poset P , then $\forall i \in B$ we have either $ImPred(i) \cap (J \setminus B) = \emptyset$ or $ImPred(i) \cap (J \setminus B)$ is an antichain. Therefore, any maximal in-module that is an antichain is simply a maximal set of elements with a common set of (immediate) successors. Similarly, any maximal out-module that is an antichain is simply a maximal set of elements with a common (immediate) predecessor set.

If $J = C_1 \cup C_2 \dots \cup C_K$ is the union of K parallel chains, then each C_i is *both* an in-module and an out-module (a bi-module) in P : if $A = C_i$, then $\forall i, j \in A$ we have $ImSuc(i) \cap (J \setminus A) = ImSuc(j) \cap (J \setminus A) = \emptyset$; and if $B = C_i$, then $\forall i, j \in B$ we have $ImPred(i) \cap (J \setminus B) = ImPred(j) \cap (J \setminus B) = \emptyset$. Furthermore, no proper nontrivial subset of a chain (i.e., not a single point) has both properties. In that case we say that each C_i is a *minimal non-trivial bi-module* in P . It is also easy to see that if the poset

$P = (J, \prec_P)$ is an in-tree, then $ImPred(i)$ is an in-module $\forall i \in J$; and if P is an out-tree, then $ImSuc(i)$ is an out-module $\forall i \in J$.

Using the above definitions, *strong precedence* relation in scheduling implies the following:

- for chains: the minimal non-trivial bi-modules (one chain from a set of parallel chains) must be scheduled in a *contiguous fashion*, i.e., every job in the chain must be scheduled *as early as possible* after the first job on the chain. If $i \prec j$ in a chain, then this implies $S_j = S_i + p_i$, where S_l is the start time for any job l and p_l is its processing time (in the case of identical parallel processors we assume w.l.o.g. that the jobs from a given bi-module are processed on the same processor);
- for in-trees: the maximal in-modules, $ImPred(i)$ for some i , must be scheduled in a *contiguous fashion* (successively), i.e., every job in the in-module must be scheduled *as early as possible* after the first job in it;
- for out-trees: the maximal out-modules, $ImSuc(i)$ for some i , must be scheduled in a contiguous fashion (successively), i.e., every job in the out-module must be scheduled *as early as possible* after the first job in it.

There is a well-established theory of ‘modular decomposition’ for posets in general and series-parallel posets in particular. This is based on the following concept of modules:

Definition 2 Let $P = (J, \prec_P)$ be a partially ordered set. $A \subseteq J$ is a **module** in P if $\forall i, j \in A$ we have $Suc(i) \cap (J \setminus A) = Suc(j) \cap (J \setminus A)$, where $Suc(j) \triangleq \{k | j \prec_P k\}$, and $Pred(i) \cap (J \setminus A) = Pred(j) \cap (J \setminus A)$, where $Pred(j) \triangleq \{k | k \prec_P j\}$.

This means that a module is a subset that ‘looks to be the same’ from the outside world, i.e., every point in the module has exactly the same predecessors, successors and incomparable elements outside of it. We note that earlier we have defined in-modules and out-modules on the transitive reduction (Hasse diagram) of the poset. Modules are usually defined on the transitive closure of the poset, but it is also possible to define them, in a more complicated way, on the Hasse diagram. Furthermore, it is easy to see that every bi-module is also a module, but the reverse is not true: E.g. any proper subchain of a chain is a module but not a bi-module.

There are polynomial time algorithms for certain scheduling objectives on large classes of (weak) precedence constraints, including series-parallel posets (see [19] for details). They are all based on the following job-module property:

Definition 3 Let $P = (J, \prec_P)$ be a partially ordered set representing the weak precedence constraints for a scheduling problem and let $A \subseteq J$ be a **module** in P . Consider an optimal schedule (sequence) σ_A of the jobs for the sub-problem induced on the set A . We say that the scheduling problem satisfies the **job-module property** if there is an optimal schedule (sequence) σ for the whole problem such that σ orders the jobs in A (i.e., their starting times) in the same sequence (relative to each other) as σ_A .

Definition 4 Series-parallel poset:

1. A poset with a single element is series-parallel;
2. If $P_1 = (J_1, \prec_{P_1})$ and $P_2 = (J_2, \prec_{P_2})$ are disjoint series parallel posets, then
 - (a) their **series composition** $P = P_1 * P_2$ is the poset on $J_1 \cup J_2$ with $i \prec_P j$ iff $((i \prec_{P_k} j \text{ for } i, j \in P_k \text{ and } k = 1, 2) \text{ or } (i \in P_1 \text{ and } j \in P_2))$;

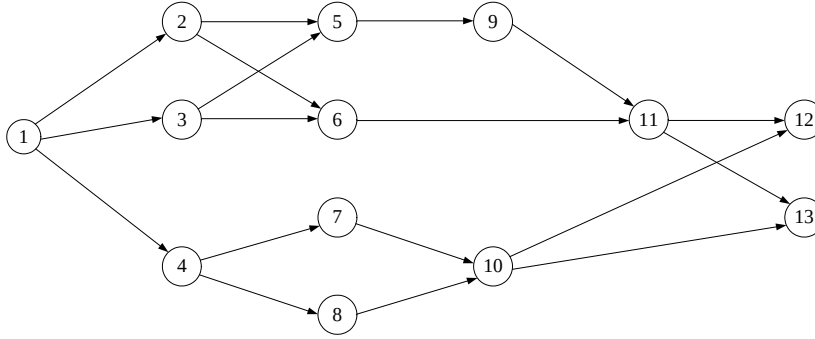


Fig. 1 A series-parallel poset

(b) and their **parallel composition** $P = P_1 + P_2$ is the poset on $J_1 \cup J_2$ with $i <_P j$ iff $((i <_{P_k} j \text{ for } i, j \in P_k \text{ and } k = 1, 2))$.

A poset is **series-parallel** if it can be built up from single elements by a sequence of series and parallel compositions.

The Hasse diagram for a series-parallel poset is shown in Figure 1.

In other words, series-parallel posets constitute the smallest class of partial orders that contains the one-element poset and is closed under series and parallel composition. As a consequence of the recursive definition, every series-parallel poset can be represented in a natural way by a binary tree, the *binary decomposition tree* (see Fig. 2). The leaves of this tree represent one-element posets, and each internal node corresponds to a partial order obtained by the series (*S*) or parallel (*P*) composition of the posets corresponding to the left and right son of the node.

It is clear that a chain on n elements can be built up by a sequence of $n - 1$ series compositions. If $P = C_1 \cup \dots \cup C_k$ is a collection of disjoint chains, then it can also be considered as $P = C_1 + \dots + C_k$, i.e., the parallel composition of these chains. Furthermore, it is easy to see that if P is an in-tree or an out-tree, then it can be built by a sequence of series and parallel compositions, such that one of the two sets in every series composition always has only a single element. This means that parallel chains, in-trees or out-trees are all special cases of series-parallel posets.

Note that P_1 and P_2 above are both modules in the above compositions. They will be referred to as **series** and **parallel modules**, respectively. The scheduling problem $1|prec|\sum w_i C_i$ ([22]), jump number ([23]), and a number of other problems (see [20]) all satisfy the job-module property. Their solution with (weak) series-parallel precedence constraints is based on iteratively sequencing optimally the sub-problems on series or parallel modules in the binary decomposition tree until the whole problem becomes completely and optimally sequenced (see [19,20]).

Let P be the series (parallel) composition of the posets P_1 and P_2 and let σ_i be an optimal sequence for the subproblem defined on $P_i, i = 1, 2$. If $P = P_1 * P_2$, then the job-module property guarantees that $\sigma = (\sigma_1, \sigma_2)$ is optimal for the problem defined

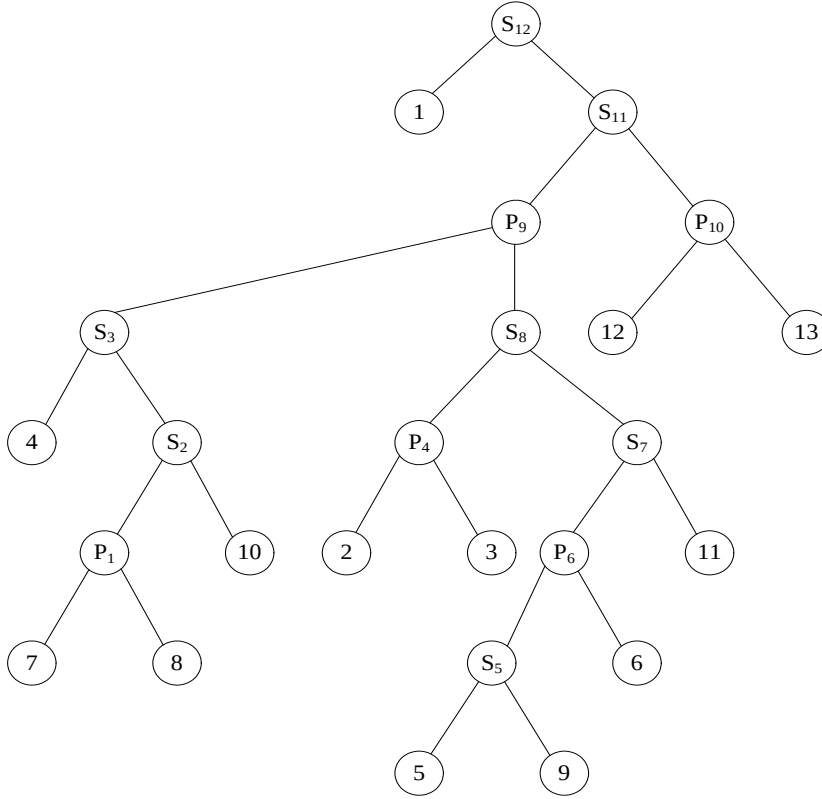


Fig. 2 The binary decomposition tree for the series-parallel poset of Fig. 1

on P . On the other hand, if $P = P_1 + P_2$, then the parallel chains corresponding to the sequences σ_1 and σ_2 have to be optimally merged into a sequence σ by some algorithm. For various implementations of this argument we refer the reader to [17, 20].

Consider the poset Z defined on four elements by the following precedence relations: $a \prec c, b \prec c$ and $b \prec d$. The following is a useful alternative characterization of series-parallel posets (see e.g. [19]).

Theorem 1 *A poset P is series-parallel if and only if it does not contain any sub-poset isomorphic to the poset Z .*

Using our earlier interpretation for strong precedence constraints, we “unify” the concept of strong precedence for both chains and trees and in addition extend it to series-parallel posets.

Definition 5 Consider a scheduling problem with $P = (J, \prec_P)$ a poset on the job set J with $\prec_P$ representing the transitive reduction of relations in P . We say that P represents **strong precedence constraints**, denoted by *strong prec*, for the problem if every schedule has to satisfy the following:

1. If $i \prec_P j$ then for the start times, we have $S_j \geq S_i + p_i$, i.e., every relation in $\prec_P$ is satisfied as a (weak) precedence constraint;
2. If P contains a parallel-chain bi-module (chain $\mathbf{C}$), then the jobs in $\mathbf{C}$ must be scheduled in a **contiguous fashion**, i.e., every job in $\mathbf{C}$ must be scheduled as early as possible after scheduling the first job in $\mathbf{C}$.
3. Consider any non-trivial maximal in-module or out-module $A \neq \emptyset$ that is not a parallel-chain bi-module and which has a non-empty successor or predecessor set, respectively. Then A must be scheduled in a **contiguous fashion**, i.e., every job in A must be scheduled as early as possible after scheduling the first job in A .

The definition implies that every feasible schedule satisfying the strong precedence constraints P would also have to be feasible for a version of the problem in which the poset P represents only weak precedence constraints. Of course, a feasible schedule for strong precedence constraints also must satisfy the additional contiguity constraints. We note that the contiguity constraints would be meaningless for the trivial modules; these were excluded from the definition in point 3. Furthermore, we require the contiguity constraint only for non-trivial in-modules with a non-empty successor set to exclude the contiguity constraint for $A = \max P$, where $\max P$ denotes the maximal elements of P . Similarly, requiring a non-empty predecessor set for out-modules excludes $A = \min P$, where $\min P$ denotes the minimal elements in P . All modules which need to satisfy the contiguity constraint can be replaced by a composite job composed of the jobs in the in-module or the out-module. We will denote this by putting these jobs between parentheses (in any sequence). If jobs in a composite job have to be processed in a prescribed sequence, then we will call such a composite job a *string* and will denote it by putting a $\rightarrow$ above this prescribed sequence of the jobs.

3 Single-machine problems

It is natural to examine how the complexity of a scheduling problem is affected if its weak precedence constraints are replaced by strong precedence. In the following discussion, we demonstrate that the relationship between the complexities of the two problems is nontrivial.

3.1 Hard problems

Dror et al. [8] proved the following theorem for regular cost functions f_j .

Theorem 2 *If the $1|\text{chain}|\sum f_j(C_j)$ scheduling problem is NP-hard, then the $1|\text{strong out} - \text{tree}|\sum f_j(C_j)$ problem is also NP-hard. If the $1|\text{chain}|\max f_j(C_j)$ scheduling problem is NP-hard, then the $1|\text{strong out} - \text{tree}|\max f_j(C_j)$ problem is also NP-hard.*

The proof is based on reducing the $1|\text{chain}|\sum f_j(C_j)$ (or $1|\text{chain}|\max f_j(C_j)$) problem to a $1|\text{strong out} - \text{tree}|\sum f_j(C_j)$ (or $1|\text{strong out} - \text{tree}|\max f_j(C_j)$) problem, where there is a one-to-one correspondence between the optimal schedules. It immediately yields the following corollary.

Corollary 1 *The $1|\text{strong out} - \text{tree}, p_j = 1|\sum U_j$ problem is NP-hard.*

i	1	2	3	4	5	6	7	8	9	10	11	12	13
w_i	1	1	3	2	7	2	10	3	1	3	4	9	1
p_i	1	4	1	3	2	8	1	5	10	7	8	2	6

Table 1 The data for an instance of $1|strong\ prec|\sum w_i C_i$

It is also shown in [8] that there are instances of $1|strong\ out - tree, p_j = 1|\sum U_j$ and $1|out - tree, p_j = 1|\sum U_j$ defined on the same job set and same out-tree poset representing the precedence constraints for which the ratio of the optimal objective function values of the two problems can be arbitrarily large. Thus adding the contiguity requirement of the strong precedence constraints can make the cost of an optimal schedule increase by an arbitrarily large amount.

3.2 Problems with strong series-parallel precedence constraints

Next we discuss how we can extend solution algorithms for scheduling problems satisfying the job-module property from weak to strong precedence constraints. For illustration we use the instance of $1|strong\ series - parallel|\sum w_i C_i$ whose precedence constraints are represented by the poset of Figure 1 and whose data is given in Table 1. (The data is the same as in the example solved by Lawler [17], allowing us to compare the optimum cost with and without strong precedence constraints.)

Consider a poset $P = (J, <_P)$ and consider the maximal in-modules in P that are not parallel-chain job modules and have non-empty successor sets. As noted earlier, each of these sets must be a maximal antichain whose elements have a common set of immediate successors. No two of these maximal in-modules can overlap by definition. Thus, these maximal in-modules together with the set $\max P$ define a unique partition of J (which can be found in polynomial time) and is denoted by $\mathcal{IN}$. This partition contains the following sets for the example poset of Figure 1: $\mathcal{IN} = \{\{1\}, \{2,3\}, \{4\}, \{5\}, \{6,9\}, \{7,8\}, \{10,11\}, \{12,13\}\}$. Next we show that the contiguity constraints on in-modules and out-modules induce certain additional weak precedence relations between some of these modules that extend the original poset P .

Extensions of type 1 (between in-modules)

Consider two in-modules, $A_1, A_2 \in \mathcal{IN} \setminus \{\max P\}$ such that there is a $k \in A_1$ and $l \in A_2$ such that $k \prec l$ in P . The fact that A_1 and A_2 must be scheduled contiguously implies that *every* job from A_1 must precede *every* job in A_2 in every feasible schedule. This implies that the weak precedence constraints $u \prec v$ can be added to P for every $u \in A_1$ and $v \in A_2$ to get an extended poset P' . This however means that the jobs in A_2 have not only the same (immediate) successors, but also the same (immediate) predecessors in the resulting extended poset P' . In other words A_2 is a job module in the poset of P' . For example, consider $A_1 = \{5\}$ and $A_2 = \{6,9\}$ in the poset of Figure 1: Since $5 \prec 9$ and $\{6,9\}$ must be scheduled contiguously, this implies that 5 must also precede 6 in any feasible schedule, so we can add the (weak precedence) relation $5 \prec 6$ to P . This results in the same predecessor set for 6 and 9, which makes $A_2 = \{6,9\}$ a job module in the extended poset.

Consider again a poset $P = (J, <_P)$ and consider the maximal out-modules that are not parallel-chain job modules and have non-empty predecessor sets. As noted earlier,

each of these sets must be a maximal antichain whose elements have a common set of immediate predecessors. Again, no two of these maximal out-modules can overlap by definition. Thus, these maximal out-modules together with the set $\min P$ also define a unique partition of J (that can be found in polynomial time) and is denoted by $\mathcal{OUT}$. For the poset example in Figure 1 this partition contains the following sets: $\mathcal{OUT} = \{\{1\}, \{2,3,4\}, \{5,6\}, \{9\}, \{7,8\}, \{10\}, \{11\}, \{12,13\}\}$.

Extensions of type 2 (between out-modules)

Consider now two out-modules, $B_1, B_2 \in \mathcal{OUT} \setminus \{\min P\}$ such that there is a $k \in B_1$ and $l \in B_2$ and also $k \prec l$ in P . The fact that B_1 and B_2 must be scheduled contiguously implies that *every* job from B_2 must follow *every* job from B_1 in every feasible schedule. This implies that the weak precedence constraints $u \prec v$ can be added to P for every $u \in B_1$ and $v \in B_2$ to get an extended poset P' . This however means that the jobs in B_1 have not only the same (immediate) predecessors, but also the same (immediate) successors in the resulting extended poset P' . In other words, B_1 is a job module in the poset of P' . For example, consider $B_1 = \{2,3,4\}$ and $B_2 = \{5,6\}$ in the poset of Figure 1: Since $2 \prec 5$, $2 \prec 6$ and $\{2,3,4\}$ must be scheduled contiguously, this implies that 4 must also precede 5 and 6 in any feasible schedule, so we can add the (weak precedence) relations $4 \prec 5$ and $4 \prec 6$ to P . Similarly, considering $\{7,8\}$ for B_2 , we can add the relations $2 \prec 7$, $2 \prec 8$, $3 \prec 7$ and $3 \prec 8$ to P . This results in having every element of B_1 precede every element of $\{5,6\} \cup \{7,8\}$, which makes all successors for the elements of B_1 common, i.e., B_1 becomes a job module in the extended poset. Similarly, looking at $B_1 = \{5,6\}$ and $B_2 = \{9\}$, the original relation $5 \prec 9$ and the contiguity requirement for B_1 imply that we can add the (weak precedence) relation $6 \prec 9$, which makes $B_1 = \{5,6\}$ a job module in the extended poset.

Extensions of type 3 (between overlapping out- and in-modules)

Consider now an $A \in \mathcal{IN} \setminus \{\max P\}$ and a $B \in \mathcal{OUT} \setminus \{\min P\}$ such that $U = A \cap B \neq \emptyset$, but $A \neq U$ and $B \neq U$. By definition both A and B are constrained to be scheduled contiguously. Moreover, every job in $A \setminus U$ must be scheduled contiguously with the jobs in U , and every job in $B \setminus U$ must also be scheduled contiguously with U . This implies that the common elements U "glue" the overlapping modules together into composite jobs, and all feasible sequences are of the form $(A \setminus U), U, (B \setminus U)$ or $(B \setminus U), U, (A \setminus U)$. Now take a job $u \in U$. We claim that if P is a series-parallel poset, then there must be an $a \in (A \setminus U)$ and $b \in (B \setminus U)$ with $b <_P a$: There cannot be an $a \in (A \setminus U)$ and $b \in (B \setminus U)$ with $a <_P b$ since this would imply $a <_P u$ (as $u \in B$ and B is an out-module), which is not possible since $u \in A$ and A is an in-module. If we had $a \parallel b$, then for any common immediate successor x of a and u , we would have $a \prec x$ and $u \prec x$, while for any common immediate predecessor y of B , we would have $y \prec b$ and $y \prec u$, and together these would imply that $\{a, y, x, b\}$ form a subposet Z , that is forbidden in a series-parallel precedence setting by Theorem 1. Thus, we must have $b <_P a$ for $a \in (A \setminus U)$ and $b \in (B \setminus U)$. This implies that any feasible sequence for the composite jobs must be of the form $((B \setminus U), U, (A \setminus U))$, and this in turn implies that every predecessor of any job in A must also precede all the jobs in B . Also any successor of any job in B must succeed every job in A . In other words, if we add these induced weak precedence constraints to P then composite job $A \cup B$ will be a job module in the extended poset. As an illustration of this point, consider in the poset of Figure 1 the sets $B = \{5,6\}$ and $A = \{6,9\}$: We have $U = A \cap B = \{6\}$, and since $5 \prec 9$, the only feasible sequence is $(B \setminus U), U, (A \setminus U) = (5, 6, 9)$. This makes the string $(5, 6, 9)$ a

job module in the poset P extended by the relation $5 \prec 6$.

Another example for overlapping in-modules and out-modules in the poset of Figure 1 is $A = \{10, 11\}$ and $B = \{10\}$. Note that $U = A \cap B = B$ in this case, thus the gluing operation adds the "new" composite job $(U, (A \setminus U))$, which of course is just A . In this case, by applying extension 3, the weak precedence relations $9 \prec 10$, $7 \prec 11$ and $8 \prec 11$ are added, thus making A a job module in the extended poset.

Algorithm *Strong SP*

INPUT : A scheduling problem that satisfies the job-module property and whose strong precedence constraints are represented by the series-parallel poset $P = (J, <_P)$.

Step 1. Identify the partitions of J into in-modules and out-modules, respectively, denoted by $\mathcal{IN}$ and $\mathcal{OUT}$.

Step 2. Extend the poset P by adding the weak precedence constraints arising from the application of all extensions of type 1 and 2. Let the resulting poset be P' .

Step 3. Contract all nontrivial in-modules in $\mathcal{IN} \setminus \{\max P\}$ and all nontrivial out-modules in $\mathcal{OUT} \setminus \{\min P\}$ into composite jobs. If P has any parallel-chain bi-modules, then contract these into composite strings.

Step 4. Extend P' further by adding the weak precedence constraints arising from the application of all extensions of type 3 and apply the gluing operation to form composite jobs on the overlapping modules. Let the resulting extended poset be P'' .

Step 5. Apply the original series-parallel algorithm for weak precedence constraints [17,20] to the scheduling problem defined by the composite jobs and weak precedence constraints P'' .

OUTPUT : An optimal schedule.

Note that a scheduling problem on independent jobs (with empty precedence constraints) typically becomes polynomially solvable when the objective function satisfies the adjacent pairwise interchange property, which allows us to define a transitive and complete binary preference relation on the jobs. This relation determines an optimal sequence for the unconstrained jobs. For more detail, we refer the reader to [20]. In the case of the $1||\sum w_i C_i$ problem, this preference relation is just the well-known weighted-shortest-processing-time (WSPT) order. This order can also be extended to subsets or sequences of jobs: For any $A \subseteq J$, we define $w(A) = \sum_{i \in A} w_i$ and $p(A) = \sum_{i \in A} p_i$. The preference order on these subsets is determined by the non-increasing WSPT ratios $w(A)/p(A)$. The polynomial time solution algorithm for the $1|series-parallel|\sum w_i C_i$ problem [17,20] repeatedly uses this preference order on sets and strings of jobs.

When applying Algorithm *Strong SP* to the example whose strong precedence constraints are represented by the poset depicted in Figure 1, we add the extensions 1-3 as discussed above. The binary decomposition tree of the resulting extended weak precedence constraints P'' is shown in Figure 3.

Explanation of the schedule generated by Algorithm *Strong SP*:

Applying the original series-parallel algorithm in Step 5, we start sequencing the job-modules with the leaves of the tree in Figure 3. The elements of the composite job

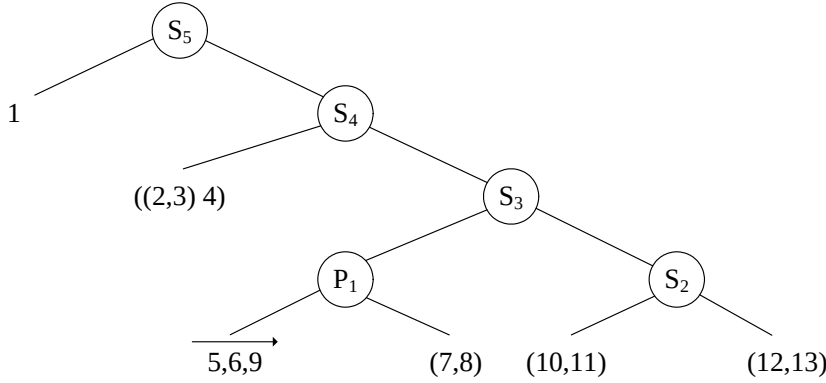


Fig. 3 The binary decomposition tree for the extended poset P'' .

$(7, 8)$ are scheduled in WSPT order resulting in the string $\overrightarrow{7, 8}$. Comparing the parallel strings $\overrightarrow{5, 6, 9}$ and $\overrightarrow{7, 8}$, $w(\overrightarrow{5, 6, 9})/p(\overrightarrow{5, 6, 9}) = 12/20$ and $w(\overrightarrow{7, 8})/p(\overrightarrow{7, 8}) = 13/6$, which means that $\overrightarrow{7, 8}$ should be scheduled before $\overrightarrow{5, 6, 9}$ in the job module corresponding to the node P_1 in the tree. The WSPT order on the elements of the composite job $(10, 11)$ is 11, 10, so it gets replaced by the string $\overrightarrow{11, 10}$. The WSPT order on the elements of $(12, 13)$ is 12, 13, so it gets replaced by the string $\overrightarrow{12, 13}$. The WSPT order on $(2, 3)$ is 3, 2, so it gets replaced by the string $\overrightarrow{3, 2}$. This string comes before 4 in the WSPT order, so the composite job $((2, 3), 4)$ is replaced by the string $\overrightarrow{3, 2, 4}$. Finally putting all strings in the order that satisfies the series compositions in the tree, we obtain the optimal schedule $(1, 3, 2, 4, 7, 8, 5, 6, 9, 11, 10, 12, 13)$ with total weighted completion time of 1228. In comparison, when P is interpreted to represent only weak precedence constraints, the optimal sequence is $(1, 4, 7, 3, 2, 5, 8, 10, 6, 9, 11, 12, 13)$ with $\sum w_i C_i = 1158$. This demonstrates that the strong precedence constraints can result in a substantial increase in the minimum cost.

Theorem 3 Consider a scheduling problem 1|strong series-parallel|f that satisfies the job-module property. Then Algorithm Strong SP finds an optimal schedule for the problem in polynomial time.

Proof We note that the cost function for the scheduling problem can be either of the type $f = \sum f_j(C_j)$ or $f = f_{\max} = \max_j f_j(C_j)$. The problem only needs to satisfy the job-module property.

Consider now a scheduling problem with strong series-parallel precedence constraints P and assume that the problem also satisfies the job-module property. Algorithm Strong SP contracts the nontrivial in-modules and out-modules into composite jobs and extends the original poset P by adding all the implied weak precedence relations between these modules to P . As discussed above, every feasible schedule must satisfy these extended weak precedence relations. The fact that composite jobs cannot be broken up automatically ensures that all contiguity constraints prescribed by the strong precedence constraints will be satisfied. The original series-parallel algorithm always finds an optimal schedule for the problem with the extended weak precedence constraints.

We mention the following immediate consequence of the preceding theorem.

Corollary 2 *The total weighted completion time problem with strong series-parallel precedence constraints, $1|strong\ series-parallel|\sum w_j C_j$, is solvable in polynomial time.*

4 Multi-machine Models

4.1 Mean Flow Time on Parallel Machines

4.1.1 The $Pm|strong\ chains|\sum C_j$ problem

We start our discussion by extending to any fixed number (m) of identical parallel machines a lemma that was proved in [6] for the two-machine ($m = 2$) case.

Lemma 1 *Consider a $Pm|strong\ chains|\sum C_j$ problem on K chains, $\mathbf{C}_1, \mathbf{C}_2, \dots, \mathbf{C}_K$, where $|\mathbf{C}_i| = n_i$, $i = 1, 2, \dots, K$, and let $p_{k,i}$ denote the processing time of the i th job in $\mathbf{C}_k$, $i = 1, 2, \dots, n_k$. This problem reduces to a $Pm||\sum w_k C_k$ problem on K compound jobs, with no precedence constraints between them, where compound job k has processing time $p_k = \sum_{j=1}^{n_k} p_{k,j}$ and weight $w_k = n_k$ for $k = 1, 2, \dots, K$.*

Proof Let C_{k,n_k} ($k = 1, 2, \dots, K$) be the completion time of the last job from chain $\mathbf{C}_k$ in a schedule. Since $\mathbf{C}_k$ is a bi-module, its jobs must be processed contiguously. Thus if $p_{k,i}$ denotes the processing time of the i th job in $\mathbf{C}_k$ for $i = 1, 2, \dots, n_k$, then its completion time must be $C_{k,i} = C_{k,n_k} - (p_{k,i+1} + \dots + p_{k,n_k})$. Thus, the total completion time of chain $\mathbf{C}_k$, denoted by TFT_k , is

$$TFT_k = n_k C_{k,n_k} - [(n_k - 1)p_{k,n_k} + (n_k - 2)p_{k,n_k-1} + \dots + p_{k,2}].$$

If we let $mft_k = [(n_k - 1)p_{k,n_k} + (n_k - 2)p_{k,n_k-1} + \dots + p_{k,2}]$, then this term is schedule-independent, and only the $n_k C_{k,n_k}$ part of TFT_k varies with the schedule. Therefore, the total completion time of a schedule can be written as

$$\sum_{k=1}^K \sum_{i=1}^{n_k} C_{k,i} = \sum_{k=1}^K TFT_k = \sum_{k=1}^K n_k C_{k,n_k} - \sum_{k=1}^K mft_k,$$

where the second sum is a schedule-independent constant. Thus the original problem is indeed equivalent to scheduling K unconstrained compound jobs with processing times $p_k = \sum_{j=1}^{n_k} p_{k,j}$ and weight $w_k = n_k$ for $k = 1, 2, \dots, K$.

The $Pm||\sum w_k C_k$ problem is known to be NP-hard even for $m = 2$ [2, 18], but the following easy-to-prove lemma is in the “folklore”.

Lemma 2 *The $Pm||\sum w_k C_k$ problem always has an optimal schedule which orders the jobs on each machine in shortest-weighted-processing-time (WSPT) order.*

Dror et al. [6] showed how to partition optimally a WSPT sequence of (compound) jobs between two machines by formulating this problem as a 0-1 program. They cited Kubiak’s dynamic programming algorithm [15], which solves this 0-1 program on K jobs in $O(K \sum w_k)$, i.e., pseudo-polynomial time. Notice, however, that Lemma 1 reduces $P2|strong\ chains|\sum C_j$ to a $P2||\sum w_k C_k$ problem on K compound jobs with weights $w_k = n_k$ for $k = 1, 2, \dots, K$. Thus, $\sum w_k = \sum n_k = n$ in this case. Combining these results, they obtain the following:

Theorem 4 *An optimal solution for the $P2|strong\ chains|\sum C_j$ problem on K chains can be found in $O(Kn)$ time.*

In contrast, we mention the following result of Du et al. [9].

Theorem 5 *The $P2|chains|\sum C_j$ problem is NP-hard in the strong sense.*

Next we show how we can generalize Theorem 4 to the case of $m > 2$ machines. We use the following dynamic programming (DP) algorithm, which essentially follows the recursive scheme originally suggested by Rothkopf [21], where it is assumed that it is possible to index (order) the jobs in such a way that the jobs assigned to a given machine can be assumed to be processed in order of their indices.

Algorithm *DP*

Input: N jobs indexed in the (WSPT) order $1, 2, \dots, N$.

Let $F_j(t_1, \dots, t_m)$ be the minimum cost of a schedule for jobs $1, 2, \dots, j$ subject to the constraint that the last job on machine i is completed at t_i for $i = 1, 2, \dots, m$. Then we have the following recursion

$$F_j(t_1, \dots, t_m) = \min_{1 \leq i \leq m} (F_{j-1}(t_1, \dots, t_i - p_j, \dots, t_m) + w_j t_i,$$

with the initial conditions

$$F_0(t_1, \dots, t_m) = \begin{cases} 0 & \text{if } t_i = 0 \text{ for all } i = 1, \dots, m \\ \infty & \text{otherwise.} \end{cases}$$

It is well-known that these recursive equations can be solved in $O(mnT^{m-1})$ time, where $T = \sum_{j=1}^N p_j$. Thus combining this with Lemma 1, we obtain the following:

Theorem 6 *The $Pm|strong\ chains|\sum C_j$ problem can be solved in $O(mnT^{m-1})$ time, where $T = \sum_{j=1}^N p_j$.*

Theorems 4, 6, and 5 suggest that mean flow time minimization on parallel machines may be somewhat easier with *strong chains* precedence constraints than just with *chains*. When the number of machines is also considered to be part of the input for the problem, however, then the situation reverses.

Theorem 7 [5] *The $P|strong\ chains, p_j = 1|\sum C_j$ problem is NP-hard in the strong sense.*

Theorem 8 [7] *The $P|chains, p_j = 1|\sum C_j$ problem can be solved in polynomial time.*

4.1.2 The $Pm|strong\ chains|\sum w_j C_j$ problem

It can be easily seen that Lemma 1 can also be extended to the weighted mean flow time problem:

Lemma 3 *Consider a $Pm|strong\ chains|\sum w_j C_j$ problem on K chains, $\mathbf{C}_1, \mathbf{C}_2, \dots, \mathbf{C}_K$, where $|\mathbf{C}_i| = n_i$ for $i = 1, 2, \dots, K$, and let $p_{k,i}$ and $w_{k,i}$ denote the processing time and weight, respectively, of the i th job in $\mathbf{C}_k$ for $i = 1, 2, \dots, n_k$. Then this problem reduces to a $Pm|\sum w_k C_k$ problem on K compound jobs, with no precedence constraints between them, where compound job k has processing time $p_k = \sum_{j=1}^{n_k} p_{k,j}$ and weight $w_k = \sum_{j=1}^{n_k} w_{k,j}$ for $k = 1, 2, \dots, K$.*

Proof Let C_{k,n_k} ($k = 1, 2, \dots, K$) be the completion time of the last job from chain $\mathbf{C}_k$ in a schedule. Since $\mathbf{C}_k$ is a bi-module, its jobs must be processed contiguously. Therefore, the completion time of the i th job in $\mathbf{C}_k$ can be written as $C_{k,i} = C_{k,n_k} - (p_{k,i+1} + \dots + p_{k,n_k})$. Thus, the total weighted completion time of chain $\mathbf{C}_k$, denoted by TWT_k , is

$$TWT_k = w_k C_{k,n_k} - [(w_k - w_{k,n_k})p_{k,n_k} + (w_k - w_{k,n_k} - w_{k,n_k-1})p_{k,n_k-1} + \dots + (w_k - \sum_{j=2}^{n_k} w_{k,j}p_{k,2})].$$

If we let $mwk_k = [(w_k - w_{k,n_k})p_{k,n_k} + (w_k - w_{k,n_k} - w_{k,n_k-1})p_{k,n_k-1} + \dots + (w_k - \sum_{j=2}^{n_k} w_{k,j}p_{k,2})]$, then this term is schedule-independent, and only the $w_k C_{k,n_k}$ part of TWT_k varies with the schedule. Therefore, the total weighted completion time of a schedule can be written as

$$\sum_{k=1}^K \sum_{i=1}^{n_k} w_{k,i} C_{k,i} = \sum_{k=1}^K TWT_k = \sum_{k=1}^K w_k C_{k,n_k} - \sum_{k=1}^K mwk_k,$$

This problem reduces to a $Pm || \sum w_k C_k$ problem on K compound jobs, with no precedence constraints between them, where compound job k has processing time $p_k = \sum_{j=1}^{n_k} p_{k,j}$ and weight $w_k = \sum_{j=1}^{n_k} w_{k,j}$ for $k = 1, 2, \dots, K$.

Lemma 3 means that we can apply the Algorithm DP even to the $Pm || \sum w_k C_k$ problem that the $Pm | strong chains | \sum w_j C_j$ problem was shown to reduce to. Thus we obtain the following theorem.

Theorem 9 *The $Pm | strong chains | \sum w_j C_j$ problem can be solved in $O(mnT^{m-1})$ time, where T is the sum of all job processing times.*

4.2 Makespan Problems on Parallel Machines

Minimizing the makespan on parallel machines is one of the most extensively studied problems in the literature. Even $P2 || C_{\max}$ is known to be NP-hard and when the number of machines is considered to be part of the input, i.e., we consider $P || C_{\max}$, the problem becomes strongly NP-hard [11]. This indicates that there is not much hope for obtaining polynomial time solutions for these problems with or without precedence constraints. The situation is somewhat better for the case of unit processing times. One of the earliest algorithms for precedence-constrained scheduling is due to T.C. Hu [13], which always schedules from the set of available jobs the one that is at the beginning of the longest current chain of unexecuted jobs (the critical path algorithm).

Theorem 10 *$P | in-tree, p_j = 1 | C_{\max}$ is solved in polynomial time by the critical path algorithm.*

An alternative polynomial time algorithm for the case of out-trees was given by Davida and Linton [4]. Garey et al. [12] studied the case when the precedence graph is the union of disjoint in-trees and out-trees (*opposing forest*). They showed that $P | opposing forest, p_j = 1 | C_{\max}$ is NP-hard, but $Pm | opposing forest, p_j = 1 | C_{\max}$ is polynomially solvable for fixed m . The first polynomial time solution for $P2 | prec, p_j =$

$1|C_{\max}$ was presented by Fujii et al. [10]. Its complexity was improved by the well-known Coffman-Graham algorithm [3] in 1972. The complexity of $P3|prec, p_j = 1|C_{\max}$ is a long-standing open question in scheduling theory.

Although relatively little is known for the above problems with strong precedence constraints, it seems that these problems may be harder than their counter-parts with (weak) precedence constraints. In contrast with Theorem 10, we mention the following result:

Theorem 11 [8, 5] *The $P|strong\ in\ -\ tree, p_j = 1|C_{\max}$ problem is NP-hard in the strong sense.*

This can be further strengthened as follows.

Theorem 12 [7] *$P|strong\ chains, p_j = 1|C_{\max}$ problem is NP-hard in the strong sense.*

The last problem becomes somewhat easier to solve on a fixed number of machines by using the above-mentioned dynamic programming algorithm first proposed by Rothkopf [21].

Theorem 13 [7] *$Pm|strong\ chains, p_j = 1|C_{\max}$ problem is solvable in $O(mn^{m+1})$ time.*

5 Summary

The notion of strong versus weak precedence in job scheduling is interesting even beyond its connection to modular decomposition for chains, trees, and series-parallel graphs. For instance, as demonstrated by Krings and Dror ([14]), it can be used for addressing the classical problem of scheduling instability in non-preemptive static list scheduling: Defining strong precedence relation in terms of precedence sequence of predetermined nested subgraphs results in a schedule that is inherently stable without the introduction of new precedence edges contrary to what was proposed prior to [14].

The main contribution of this paper lies primarily in its consistent extension of the notion of strong precedence with respect to chains, trees, and series parallel graphs. The construction of in-module, out-module, and bi-module structures, defined with respect to the transitive reduction of a poset P , allows a consistent definition of strong precedence based on modular decomposition for posets. The majority of new computational complexity results presented in this paper is the consequence of the strong precedence extension for series parallel graphs. For instance, we show, among a number of other results, that the problem $1|strong\text{-}series\ parallel\ |\sum w_j C_j$ is solvable in polynomial time. In addition, we verified the complexity results for the ‘old’ definition of strong precedence for chains and trees ([6],[7],[8]) with respect to the present ‘modular’ definition of the weak/strong precedence distinction.

Acknowledgements This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant No. OG1798-08.

References

1. A.A. Assad, M.O. Ball and R.W. Dahl, Sleeping beauties: scheduling the production of mattresses with sequence-dependent set ups, *Proceeding, 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation*, Paris, France, 229-239 (1995).
2. J.L. Bruno, E.G. Coffman and J.R. Sethi, Scheduling independent tasks to reduce mean finishing time, *Comm. ACM* **17**, 382-387 (1974).
3. E.G. Coffman and R.L. Graham, Optimal scheduling of two-processor systems, *Acta Informatica* **1**, 200-213 (1972).
4. G.I. Davida and D.J. Linton, A new algorithm for the scheduling of tree structured tasks, *Proc. Conf. Inform. Sci. and Syst.*, Baltimore, MD, USA, 543-548 (1976).
5. M. Dror, Chains and trees: 'Strong' - 'weak' order in job scheduling, *Order*, **14**, 211-228 (1997).
6. M. Dror, W. Kubiak and P. Dell'Ollmo, Scheduling chains to minimize mean flow time, *Inform. Proc. Letters*, **61**, 297-301 (1997a).
7. M. Dror, W. Kubiak and P. Dell'Ollmo, 'Strong' 'weak' chain constrained scheduling, *Ricerca Operativa*, **83**, 35-49 (1997b).
8. M. Dror, W. Kubiak and J. Y.-T. Leung, Tree precedence in scheduling: The strong-weak condition, *Inform. Proc. Letters*, **71**, 127-134 (1999).
9. J. Du, J. Y.-T. Leung and C.-H. Lin, Scheduling chain-structured tasks to minimize makespan and mean flow time, *Inform. and Comput.*, **92**, 219-236 (1991).
10. M. Fujii, T. Kasami and K. Ninomiya, Optimal sequencing of two equivalent processors, *SIAM J. Appl. Math.* **17**, 784-789; Erratum *SIAM J. Appl. Math.* **20**, 141 (1969, 1971).
11. M.R. Garey and D.S. Johnson, Strong NP-completeness results: motivation, examples and implications, *J. Assoc. Comput. Mach.*, **24**, 499-508 (1978).
12. M.R. Garey, D.S. Johnson, R.E. Tarjan and M. Yannakakis, Scheduling opposing forests, *SIAM J. Algebraic and Discrete Methods*, **4**, 72-93 (1983).
13. T.C. Hu, Parallel sequencing and assembly line problems, *Operations Research*, **9**, 841-848 (1961).
14. A.W. Krings and M. Dror, Real-time dispatching: Scheduling stability and precedence, *I.J of Foundations of Computer Science*, **10**, 313-327 (1999).
15. W. Kubiak, New results on the completion time variance minimization, *Disc. Appl. Math.*, **58**, 157-168 (1995).
16. E.L. Lawler, Optimal sequencing of a single machine subject to precedence constraints, *Management Sci.*, **19**, 544-546 (1973).
17. E.L. Lawler, Sequencing jobs to minimize total weighted completion time subject to precedence constraints, *Ann. Disc. Math.*, **2**, 75-90 (1978).
18. J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, Complexity of machine scheduling problems, *Ann. Disc. Math.*, **1**, 343-362 (1977).
19. R.H. Möhring, Computationally tractable classes of ordered sets, in *Algorithms and Order*, (I. Rival ed.), Kluwer Publ. Dordrecht, The Netherlands (1989).
20. E.L. Monma and J.B. Sidney, Sequencing with series-parallel precedence constraints, *Mathematics of Oper. Res.*, **3**, 215-224 (1979).
21. M.H. Rothkopf, Scheduling independent tasks on parallel processors, *Management Sci.*, **12**, 437-447 (1966).
22. J.B. Sidney, Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs, *Oper. Res.*, **23**, 283-298 (1975).
23. G. Steiner, On finding the jump number by substitution decomposition, *Order*, **2**, 9-23 (1985).

Case studies of successful train crew scheduling optimization

Raymond S K Kwan

Abstract UK has a large and complex passenger rail network divided into a number of franchises. Crew scheduling, one of the last stages of operations planning before services go live, is mission critical to the train operating companies, which would feel the pain of manual scheduling. As far as the author is aware, attempts by these companies to adopt an automatic optimizing train crew scheduling system were unsuccessful except with TrainTRACS. After the first adoption of TrainTRACS by ScotRail in 2003, the author has founded the University of Leeds spin-out company Tracsis to commercialise TrainTRACS fully and to expand to other operations planning functions. Since then, TrainTRACS has gained wide acceptance by the UK rail industry. This paper discusses the major factors behind the achievements of TrainTRACS drawing from practical experience of development and interaction with the industry over many years. A couple of selected case studies will be presented in the discussion.

1. Introduction

In UK resource scheduling in advance of a new set of train timetables, usually for the next six months, is called 'long term planning' (LTP). In contrast, 'short term planning' (STP) deals with temporary network disruptions such as those caused by engineering work and timetable changes for public events such as football matches. This paper concerns mainly LTP train crew scheduling.

LTP train crew scheduling produces crew shifts on a daily basis. Across the week, individual crew members would need rest days and may have to attend training events. Furthermore provisions for sickness, annual leaves and spare coverage all add to the total work force. Therefore the actual number of crew members employed is a multiple of the maximum number of shifts on any day of the week. For example, for train drivers this multiplier is about 2.0. Total train crew cost also has to account for several categories of crews such as guards, conductors, and catering staff. Rail transport incurs substantial operating costs in network infrastructure, rolling stocks, energy and train crews. In UK, train crews account for about 20 – 25% of the total operating cost. A small percentage reduction of LTP daily shifts would translate into multi-million Pounds annual savings.

Train operators would also be motivated to seek the most efficient crew schedules so as to afford ploughing back some slacks into the schedules for robustness against crews being caught up by delays. Moreover, competitive tendering of the operating franchises in the privatised UK rail industry has forced the train companies to be ever more conscious of optimizing their operations – even before a franchise has been won.

Under this backdrop of motivation to optimize, train companies are under immense pressure to automate train crew scheduling. It is one of the last stages of resource planning after timetables planning and rolling stock scheduling. As it cannot start until the preceding processes have been completed and the timetables and rolling stock schedules have been frozen, planners are usually pressurized by very tight deadlines for the production of crew schedules before the operation goes live. Automation would speed up schedule production freeing up valuable time for the planners to think strategically and to explore a fuller range of what-ifs.

For the above reasons, train operating companies (TOC) would very much want to use automatic optimizing train crew scheduling systems. Indeed, there has been major piloting use of such software systems in

the last two decades by UK train operating companies. As far as the author is aware, all except TrainTRACS have been abandoned after a short time. ScotRail was the first TOC to use TrainTRACS in house in 2003 and they have continually been an active user to date. Currently, the following TOCs are also licensed to use TrainTRACS in-house: Northern Rail, Arriva Trains Wales, Southern Railway, National Express East Coast, Virgin West Coast, Arriva CrossCountry, London & South Eastern Railway, London Midland, and London Overground.

Apart from in-house use, franchise bidding teams and consultants have also been using TrainTRACS in the last few years in UK, Australia, and Northern Europe. Increasingly, franchise bidders are looking for hard evidence that if they won, they will be able to operate as planned. TrainTRACS as a production tool is therefore used instead of simple estimation methods.

Section 2 first introduces the train crew scheduling problem and TrainTRACS. One of the key research issues in designing automatic optimization systems is the goal of closing the gap between theory and practice (Burke et al 2001). TrainTRACS evidently has been bringing automatic train crew scheduling across to the practitioners and gaining wide acceptance in the UK. This paper examines some of the factors behind its achievements in section 3, some specific cases drawn from experience with the TOCs will be presented for illustration. Finally, conclusions and remarks are given in section 4.

2. LTP train crew scheduling

Bus and train crew scheduling are a well-known class of NP-hard problems, a general discussion is given by (Kwan 2004). Although the airline crew scheduling problem may be formulated similarly at the high level (Barnhart et al. 1998; Hoffman and Padberg 1993), it is usually considered distinct from those for bus and train because of the many airline specific operational practices. Bus crew scheduling problems (Wren 1981; Wren and Rousseau 1995), which have attracted early research focus, are generally not as difficult as those for train in terms of problem size and complexity. It was only from the 1990's that there has been gradually more research on train crew scheduling (Capara et al. 1997; Kroon and Fischetti 2001; Abbink et al 2005; Kwan and Kwan 2007, Huisman 2007).

Train crew scheduling is basically partitioning of the schedule of each train into segments, which are permuted and recombined with breaks and other crew activities inserted to form crew shifts. Fig.1 shows an example of part of a train schedule being covered by 2 crew shifts. It is obvious that numerous candidate shifts are possible.

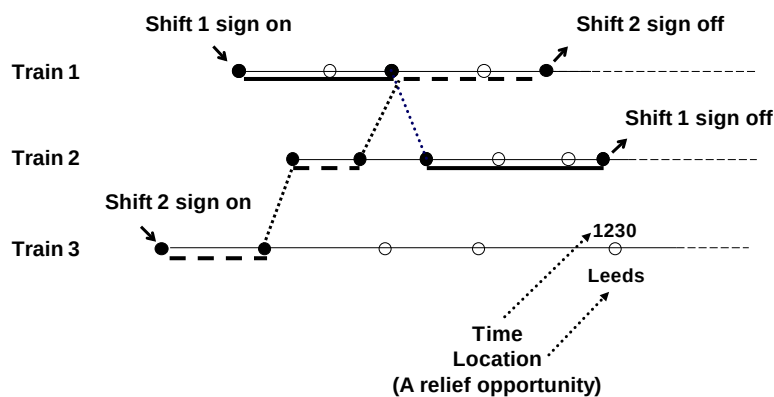


Fig.1 Train work and crew shifts

In Fig.2, the optimization problem can be formulated mathematically as an integer linear programme (ILP) based on set covering (Smith and Wren 1988).

$$\begin{aligned}
 &\text{Minimise} && W_1 \sum_{j=1}^n c_j x_j + W_2 \sum_{j=1}^n x_j \\
 &\text{Subject to} && \\
 &&& \sum_{j=1}^n a_{ij} x_j \geq 1 \quad i = 1, 2, \dots, m \\
 &&& x_j = 0 \text{ or } 1 \quad j = 1, 2, \dots, n
 \end{aligned}$$

where

n is the number of candidate shifts

m is the number of work pieces

x_j is a shift variable, $x_j = \begin{cases} 1 & \text{if shift } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$

c_j is the cost of shift j

$a_{ij} = \begin{cases} 1 & \text{if work piece } i \text{ is covered by shift } j \\ 0 & \text{otherwise} \end{cases}$

W_1 and W_2 are weight constants

Fig.2 Set covering ILP for train crew scheduling

In this section, the problem is further illustrated through a discussion of the input and output of a solution process in practice.

Train units are basic units of rolling stock that would not be broken down into any smaller sub-units during the operational day, even though a train unit may consist of multiple (e.g. 2, 4, 8, etc.) carriages. However, it is common for train units to be coupled together for parts or all of their journeys. Train units are usually scheduled before crew schedules are compiled to cover them. The schedule of a train unit is known as a ‘unit diagram’, an example of which is shown in Fig.3.

In Fig.3, the main body of a train unit diagram describes timetabled journeys that the unit serves, e.g. from Ebne at 14.55 to Vic at 16.31, via E Croy, is identified by the train ID (also known as a ‘headcode’) 1F37. The header of the unit diagram indicates that it is operated on a Sunday (“Su”). The other important information in a unit diagram concerns coupling (indicated by “ATTACH” or “ATTTT”) and decoupling (indicated by “DETACH” or “DETTT”), e.g. another unit numbered 117 is coupled with it at Ebne and departs at 09.55, which is later decoupled after arriving at Ebne again at 13.19. In scheduling train drivers, when multiple units are coupled, it is obvious that the crew scheduling system should only use one driver for the whole coupled train formation. For other crew types, e.g. Train Manager, when units are coupled, each unit might require its own crew because it might not be possible for the crew to get through the coupled units.

Whilst wheel turning work forms the bulk of a crew schedule, some non-wheel-turning work not shown in a unit diagram may also have to be done by a train crew. Such work has a significant impact on scheduling because often it could not be pre-determined, but is inserted as required dictated by the scheduling rules. For example, when a train unit is stationary at a platform, its engine may be left running if the driver remains onboard. However, the schedule may allow the driver to be relieved by a new driver. In that case, the relieved driver has to immobilize the engine before leaving, and the new driver may arrive some time after the other driver has left, and then he has to mobilize the engine again. Both immobilization and mobilization tasks have to

be given extra time allowances to the driver on top of the train arrival and departure times. Other types of idle time allowance may also have to be inserted dynamically to build robustness into the crew schedule. For example, if a train crew has to travel as a passenger, he would have to be scheduled to wait for the departing train rather than to be absolutely just-in-time.

DIAGRAM: BB238		Su	Fleet	377/1-12
Start Date: 01/12/2012		End Date : 31/05/2013		
Bton LWS		06+37	5F40	116(1)\117(2)\238(3)
Brighton	06+52			DETACH
Brighton		07+02	5F40	
Lewes	07+14	07.18	2F40	
Brighton	07.34	07.43	2D11	
Ore	09.00	09+02	5D11	
Ore Up Sd	09+04	09+10	5F27	
Ore	09+12	09.14	1F27	
Ebne	09.48			ATTTT
Ebne		09.55	1F27	238(1)\117(2)
E Croy	11.09	11.10	1F27	238(1)\117(2)
Vic	11.31	11.47	1F18	117(1)\238(2)
Hay Hth	12.40	12.41	1F18	117(1)\238(2)
Ebne	13.19			DETTT
Ebne		13.26	1F18	
Ore	14.00	14+02	5F18	
Ore Up Sd	14+04	14+10	5F37	
Ore	14+12	14.14	1F37	
Ebne	14.48			ATTTT
Ebne		14.55	1F37	238(1)\117(2)
E Croy	16.09	16.10	1F37	238(1)\117(2)
Vic	16.31	16.47	1F28	117(1)\238(2)
Hay Hth	17.40	17.41	1F28	117(1)\238(2)
Ebne	18.19	18.26	1F28	238(1)\117(2)
Ore	19.00	19+02	5F28	238(1)\117(2)
Ore Up Sd	19+04	19+10	5F51	117(1)\238(2)
Ore	19+12	19.14	1F51	117(1)\238(2)
Ebne	19.48	19.55	1F51	238(1)\117(2)
E Croy	21.09	21.10	1F51	238(1)\117(2)
Vic	21.31	21.47	1F48	117(1)\238(2)
Hay Hth	22.40	22.41	1F48	117(1)\238(2)
Ebne	23.19	23.26	1F48	238(1)\117(2)
Hastings	23.59	00+08	5F48	117(1)\238(2)
Ebne	00+30			CET
				WASH
Ebne		01+10	5F48	238(1)\117(2)
Ebne	02+18			STABLD

Fig.3 An example train unit diagram

Although there are different types of train crew together in a team on the day of operation, team work constraints do not have to be considered in LTP and each type has differing labour rule parameter values. For example, both drivers and catering staff are subject to a maximum time between signing on and signing off (called 'spreadover'), but whereas the maximum spreadover for a driver may be 9:30, that for a catering staff may be 12:30. Hence, each type of train crew is scheduled separately. Where more than one members are required onboard for a type of crew, they would be scheduled as a single unit (i.e. each notional shift is applicable to however many crew members of the same type are required).

A train crew shift is more commonly called either a 'turn' or a 'crew diagram' in the railway industry. An example of a train crew diagram is shown in Fig.4.

Given a set of train unit diagrams, a solution crew schedule consists of a set of legal shifts that together provides the particular type of crew resource required for its complete operation. The construction of a shift must conform to the union agreements, which describe generally the working conditions of the crew. Within a

shift, a crew is usually given a mealbreak (known as a 'Physical Needs Break' (PNB) in UK railway terms). Often, a shift can contain up to three mealbreaks which must start and end within specific hours, relative to the signing on time, as laid down in the union agreements. A main mealbreak is the one which divides the shift into two parts in similar proportion. A stretch is the period from the start of a shift to the start of the main mealbreak, or from the end of the main mealbreak to the end of the shift. A shift often contains two or more spells of work. A spell is a period of time a crew works continuously on a train. The time required for the crew to travel as passenger is usually excluded from a spell. In Fig.4, the main mealbreak is the period from 15:58 to 16:37 at Edinburgh where the driver is given enough walking time allowance between the platform and the canteen plus at least 15 minutes eating time. The shift has three mealbreaks, two stretches and four spells.

Not all the places where a train stops in train unit diagrams are feasible places for crew relief. A place where the train crew can be relieved is known as a 'relief point'. In Fig.4, Edinburgh is a relief point but Dunblane and North Berwick are not. The times when a train unit passes these points are called 'relief times'; a relief time at a particular relief point constitutes a 'relief opportunity'. Many relief opportunities are not used and may not be shown in the crew diagrams. For example, train CK207 in Fig.4 passes through Stirling at 19:37 and 20:26, and relief opportunities exist then. The shift shown does not use these opportunities, but they might be used in other possible solutions. When a relief opportunity is chosen for crew relief, the crew onboard will leave the train (then takes a mealbreak, finishes the shift or join another train), and another crew (which may have just started its shift or have finished a break) would take over the operation of the train. Usually, when a train arrives at a point, there is a time gap before it departs. This creates a complication as there are two relief times at one location. The arrival and departure times may or may not be close together. In this case, the relief opportunity contains two different relief times. A crew can effectively relieve another crew at any time which falls within the time gap, and this leads to a window of relief opportunities (Laplagne 2008).

2.1 TrainTRACS

TrainTRACS is based on the Generate and Select (GaS) approach (Kwan 2004). In the Generate phase, GaS constructs a pool of very many legal candidate shifts controlled by soft constraint parameters and heuristics (e.g. only allow up to three breaks in a shift). In the Select phase, a solution schedule is formed by selecting a combination of candidate shifts that covers all the work best satisfying the objectives and any additional constraints at the schedule level. The set covering model naturally fits well for the selection process with GaS. Set covering is one of the most studied NP-hard problems (Garey and Johnson 1979, Chvátal 1979). Given a ground set U of m elements (pieces of work from the train unit diagrams), and a weight (labour cost) for each set (a shift), the goal is to cover U with the smallest possible number of sets. In the case of train crew scheduling, there is the additional objective of minimising the total weight. TrainTRACS uses the ILP (see Fig.2) technique for the set covering selection process (Smith and Wren 1998; Willers et al. 1995). One advantage of the GaS approach is that optimization is mainly performed at the Select phase, which can employ a largely domain independent solver taking advantage of advances in generic solution techniques, e.g. column generation (Fores et al 1998) and hybridised search (Kwan and Kwan 2007) are used in TrainTRACS.

TrainTRACS was originated from University of Leeds, where research in transport scheduling dates back to the early 1960s. General presentations of the system are given in (Wren et al. 2003; Fores et al. 2002). The company Tracsis was founded in 2004 to fully commercialise TrainTRACS. In 2004 and 2005, several major TOCs have commissioned pilot trials using TrainTRACS for part of their current operations. After the pilot trials, the TOCs involved have all adopted TrainTRACS for regular use in-house. More similar pilot trials and subsequent adoptions by other TOCs followed. Some transport company groups have also used TrainTRACS to support their franchise bids.

<u>Driver</u>	<u>No.</u>	<u>Train No.</u>	<u>Task</u>			
On	-	BU2388	HA403	MOB		
Off	-	11:50		Edinburgh	12:10	2L06
Hours	-	21:30		Bathgate	12:38	12:48 2F36
Days	-	09:40		Edinburgh	13:14	
		SUN		IMMOB		
				mealbreak (min. 25 mins)		
			HA409	MOB		
				Edinburgh	14:15	5L28
				Inverkeithing	14:39	14:41 5L28
				Inkeithing N Jn	14:43	15:28 5L28
				Inverkeithing	15:30	15:32 2L19
				Edinburgh	15:58	
				IMMOB		
				mealbreak (min. 15 mins)		
			GW161	MOB		
				Edinburgh	16:37	2B31
				North Berwick	17:10	17:20 2G31
				Edinburgh	17:53	
				IMMOB		
				mealbreak (min. 20 mins)		
			CK207	MOB		
				Edinburgh	18:48	2R42
				Dunblane	19:48	20:06 5R42
				Dunblane Sig Box	20:07	20:14 5R31
				Dunblane	20:15	20:18 2R56
				Edinburgh	21:18	
				IMMOB		

Fig.4 An example train crew (driver) diagram

3. Successful train crew scheduling optimization

TrainTRACS has persisted and evolved over a large number of years, not being exhaustive the main factors are:

- Mathematical engine consistently yields efficient solutions
- Solutions produced are readily operable
- Algorithmic approach suits long term development in which application domain knowledge could be accumulated incrementally

In this section, these factors are examined and a couple of case studies working with UK TOCs will be presented.

3.1 Optimizing capability

To most train planners, next to validity what they look for is efficiency in the schedules. Pilot trials of TrainTRACS have demonstrated to the TOCs very significant efficiency savings. Because of commercial sensitivity, TOCs usually would not be making public the saving figures. They would also, sometimes quite rightly, argue that there are other factors than using TrainTRACS in achieving the savings. In some cases, the TOC may decide not to implement reduction of their crews employed immediately (therefore ‘no saving’!), but to use the excess crews to increase the number of spare cover turns. In any case, the UK train industry is very self-referencing and the reputation of the tools and systems they use does get widely spread. In the following, a very recent case study is presented in which the schedule efficiency achieved by TrainTRACS was quite remarkable.

Case Study: Virgin West Coast

Virgin West Coast operates an inter-city network from London along the West Coast up to Scotland serving the major cities of Birmingham, Liverpool, Manchester and Glasgow. The West Coast Route Modernisation project (Modern Railway 2009), which has spanned over many years, was being completed by the end of 2008. The network capacity has been increased and trains are now able to pass through some previously bottlenecks at high speeds. As a result, journey times are shortened and Virgin was committed to a new set of timetables with vastly increased services (“Virgin High Frequency” (VHF) services). The number of weekday services was increased from 252 to 335, Saturday services was increased from 230 to 300, and Sunday services goes from 155 to 230 trains (Modern Railways 2008). Most notably, with a train every 20 minutes between London, Birmingham and Manchester in both directions it has become the most frequent long-distance inter-city service in Europe.

Around late 2007, Virgin began crew planning for the VHF timetable. With a 32% increase in services, even though they could manage (because of shortened journey times and improved maintenance facilities) with the existing fleet of trains, Virgin expected an increase in crew size. However, there was an obvious need to try minimizing the increase. From another franchise Virgin Cross Country, Virgin had prior experience of trialling and adoption of TrainTRACS. (When Cross Country’s franchise expired, Arriva won the new franchise and they continued using TrainTRACS.) Hence, a pilot trial was carried out between January and March 2008 using TrainTRACS for planning the crew schedules for the VHS services. The results of the pilot trial was very carefully scrutinised and management was impressed and confident that TrainTRACS was the tool they needed. Virgin quickly adopted TrainTRACS and used it in-house for planning the VHF crew schedules in full scale.

The Virgin onboard train crews include drivers, train managers and catering staff. Part of the train crew planning involves negotiations on changing some of the labour rules. Several key changes were accepted by the train crews. At the end only half a dozen more train managers were needed (Modern Railways 2009) and there was no increase in the other categories - out of a total of over 1600 onboard train crews, for the whole VHF crew schedule. The VHF timetable, covered by crew schedules produced by TrainTRACS, went live in December 2008.

The Virgin West Coast problem is highly constrained. As an example of schedule efficiency achieved by TrainTRACS, one of the Virgin driver labour rules is so-called “no double tripping”, meaning that a driver cannot be scheduled to drive the same route section in the same direction for more than once within a shift. Before the VHF timetable, many driver shifts would only cover one major city destination from London between breaks, and the breaks at London were typically very long (up to about 3 hours). The new driver schedules still adhere to the “no double tripping” rule, but some “triangular working” between London and two major city destinations would be covered, resulting in the removal of long breaks.

3.2 Solution operability

For a crew schedule to be operable means that basically

- No train service or empty movements would be without crew cover

- Crew members are qualified for the work they are scheduled to do
- Crew members can get to where they need to be on time

Achieving total operability in its solutions is the ethos in the development of TrainTRACS. For some other scheduling problems, maybe for example an 80% valid solution would be acceptable and the user would either not to operate or manually making good the remaining 20%. Train services cannot be left uncovered and manual adjustments would be difficult due to the combinatorial nature of the schedule.

The TrainTRACS strategy has two main folds, it does not allow infeasible shifts at any stage of its algorithm, and it provides comprehensive features modelling the operations at the required level of details. Examples of two key features are:

- Crew travel as passengers. When a train crew is scheduled to cover a train at a distant location, we have to ensure precisely that there is a means of transport that will get the train crew there on time. For the scheduling of urban bus drivers, a constant time allowance would usually be adequate for a bus driver to travel from anywhere to anywhere within the bus network. Because of the wide area of network coverage, train crews often have to catch the train themselves and travel according to the timings in the train timetables; using just a constant travel time allowance often would not yield valid solutions. TrainTRACS computes the shortest paths using the timetabled transport and walking.
- Route and traction knowledge. Each train work piece has a set of route and traction skill requirements for the crew covering it. The TrainTRACS model includes lookup tables of which set of train crews, usually in groups within specific depots, knows which routes and traction types. When TrainTRACS forms crew shifts, only work pieces with compatible route and traction knowledge are allowed in the same shift.

TrainTRACS also has numerous other features for modelling labour rules and time allowances for individual shifts, and for modelling side constraints on parts of or the whole of the schedule. In the following, a case study of some experience with Southern Railway is presented to highlight a special scenario requiring accurate modelling of their operations.

Case Study: Southern Railway

Southern covers many large densely populated areas in south England from London to the coasts, e.g. Southampton, Portsmouth, Brighton and Eastbourne. Its services are of mainly intensive urban commuting type. After a brief pilot trial, Southern adopted TrainTRACS for in-house use in 2006. Their network was amongst the largest and most difficult to be tackled in a feasibility trial for TrainTRACS. The feasibility trial was carried out on the Saturday services over the whole network. Because of the intensive commuter services, there are many relatively short interval stops giving rise to very many relief opportunities and small work pieces. Southern has many crew depots scattered around the network, and the crews are subdivided into groups called 'links' within each depot. Drivers from each link may have knowledge of different combinations of routes posing a complex compatibility structure for the scheduling process. There are targets on the minimum and maximum number of turns for each depot link. Southern has a practice called 'full crew working', which means that onboard crew members will be scheduled as a team for the whole day. Full crew working has the advantage that working as a team generally increases schedule robustness and is usually welcomed by the crew. In this pilot, the full crew consists of a driver and a conductor. Usually, conductor problems are smaller than the driver problems because conductors do not have extra train work such as preparation, disposal, etc. Also, the labour rules for conductors and drivers are different. For example, the maximum turn length is usually higher for conductors than for drivers.

At Southern, train drivers have to engage in some non-driving tasks notably the coupling and de-coupling of train units. Exactly how the drivers are scheduled to share the driving and non-driving work at the coupling/de-coupling instances impinges on the operability of the schedule. A specific and frequently occurring scenario is presented below for illustration.

Basic scenario

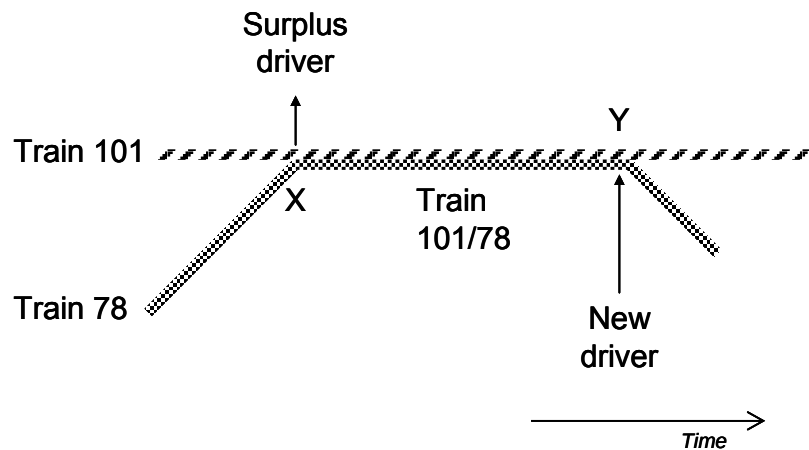


Fig.5a Coupling of two trains

Fig.5a illustrates the attachment of train unit 78 to the rear of train unit 101 after they have arrived at a certain station X. The coupled train 101/78 only needs one driver making the other driver not needed at X. Normally, the driver at the rear would get off train, perform the attachment and let the driver on the front train to continue the train journey. Otherwise, the driver on the front train has to walk to the rear, does the attachment, and walk back to the front before setting off again, which may require more time than the train timetable allows. Therefore TrainTRACS applies the rule that “Driver on the front unit stays on” and pre-processes to remove the coupled work from all but the front train unit, which is illustrated in Fig.5b. Later on, the coupled units are detached at Y and go separate directions. This time one more driver is needed, who would be waiting at Y to do the detachment and then drives the rear unit. The pre-processed train unit diagram 78 becomes two segments separated by a large time gap, which would force its driver to be relieved at the start of the gap and another driver to get on after the end of the gap.

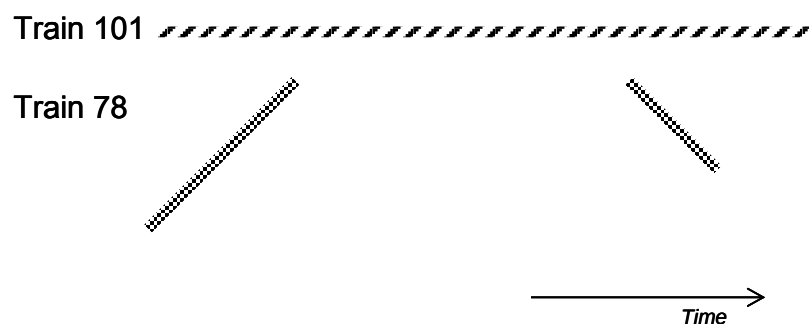


Fig.5b Pre-processed train work, basic scenario

Complex scenario with train reversing

Fig.6a illustrates a frequently occurring scenario: suppose now the coupled train stops at terminal Z, the driver walks to the other end of the train and continues the train journey in the reverse direction.

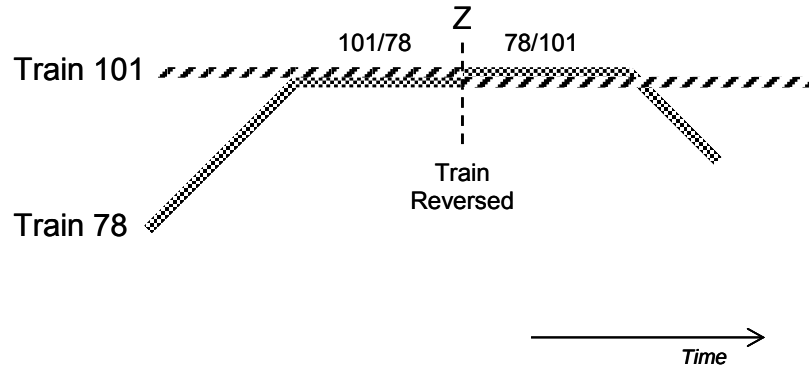


Fig.6a. Coupled train reversed direction

Train 101 becomes at the rear instead of at the front and vice versa for train 78. Applying the pre-processing rule used in the basic scenario, Fig.6b shows the results of removing train work in which the train unit is not at the front.

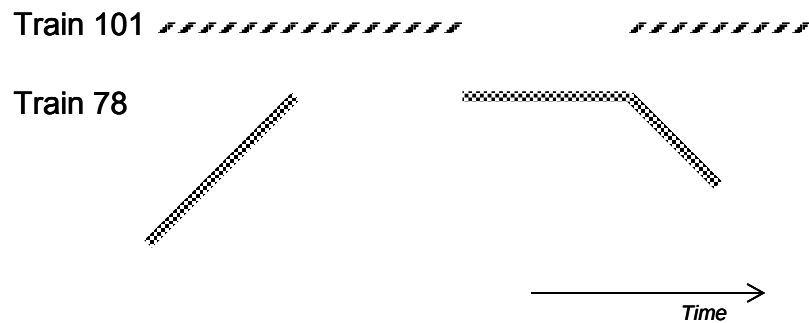


Fig.6b. Rear coupled train work removed

Fig.6b is unsatisfactory because:

- The train work is more fragmented (into 4 segments) than the scenario shown in Fig.5b (3 segments). In general, dividing the train work into more segments would result in more short pieces of continuous driving work making the schedule less efficient.
- Each pre-processed train unit has a large time gap between its two segments, forcing a crew relief at Z, but Z might not be a suitable location for changing crews.

The planners at Southern observed that the resulting schedule is exhibiting a style of shift composition quite different from how they would operate in practice. The problem was intensively studied to identify all the variants of the scenario and the exact rules to be applied resulting in a different pre-processing method rearranging ('reshaping') the train work between the unit diagrams involved as illustrated in Fig.6c. Since the driver has repositioned to the other end of the train before the coupled train reverses direction, the driver on train 101 is at the front of the train all the time. The coupled work is therefore completely removed from train 78. Moreover, at the end of the coupled work, the driver is actually on the original train 78 because of the train reversal. Therefore the final leg of un-coupled work on train 78 is also appended to train 101 - the original final leg of which will now be at the rear and therefore detached and given a different train unit number '101a'. The reshaped unit diagrams are only for use by crew scheduling and no longer preserves the true identity of the rolling stock. By this method, the number of train work segments for crew scheduling is minimized.

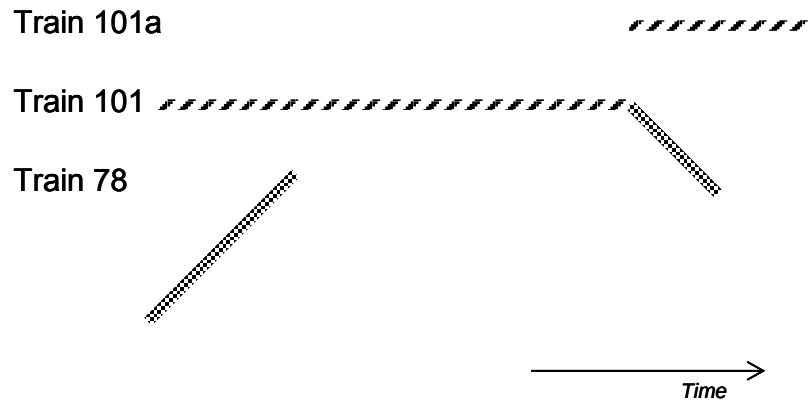


Fig.6c. Rearranged train work

The scenarios are simplified in Figures 5-6 for illustration purposes. In real life, multiple train units may be involved in numerous different combinations of coupling during a day's operation. Examples of many instances of reversal of coupled train units can be found in Fig.3, e.g. 16.31 at Vic and 18.19 at Ebne. Fig.7a shows diagrammatically an extract of a part of a train network in Southern. The operations are evidently complex with respect to how train units are coupled. For example, focusing on train unit 104, in the earlier part of the day it is coupled to the rear of unit 103; the coupled train is further attached to the rear of unit 133; the three-unit formation reverses direction of travel so that 104 becomes at the front; unit 104 is detached on its own before being attached to the rear of the coupled formation 129/137; reverses direction, detached and later attached to unit 133 again. TrainTRACS extends the 'reshaping' process discussed above transforming the train work from Fig.7a into Fig.7b with continuous segments of reshaped train work shown in textured lines. In the process, various additional tasks and their time allowances are also computed, which are needed for compiling the crew schedule. Examples are: PB (prepared from berth), PC (partial prepare), TC, B (berth), AB (attach and berth multiple units), AP (attach portion), APR (attach portion rear), DWFP (detach and work front portion) etc... Each type of task attracts a slightly different time allowance for the driver - sometimes dependent on the length of the train, sometimes dependent on whether the train is at a siding or at a platform.

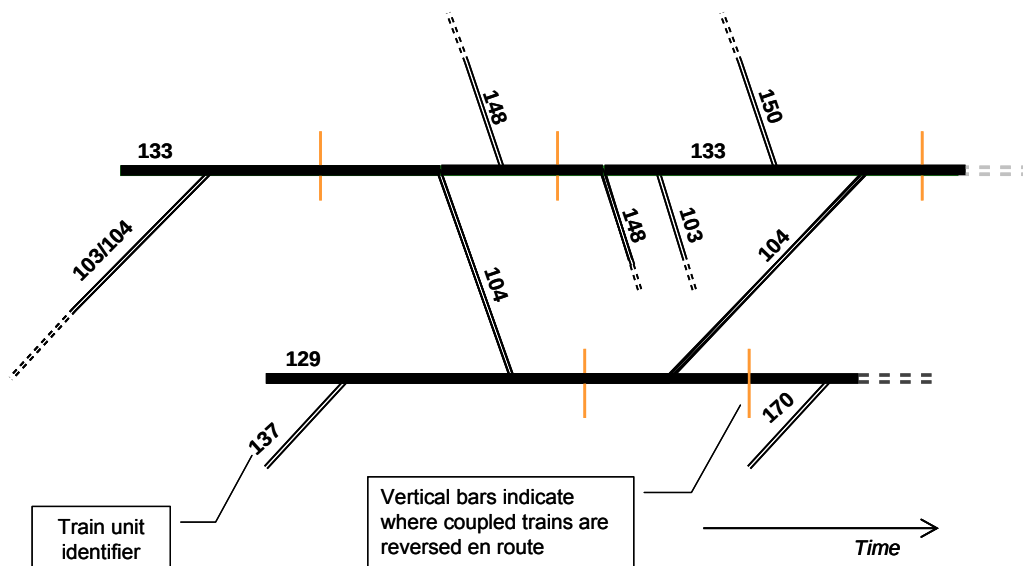
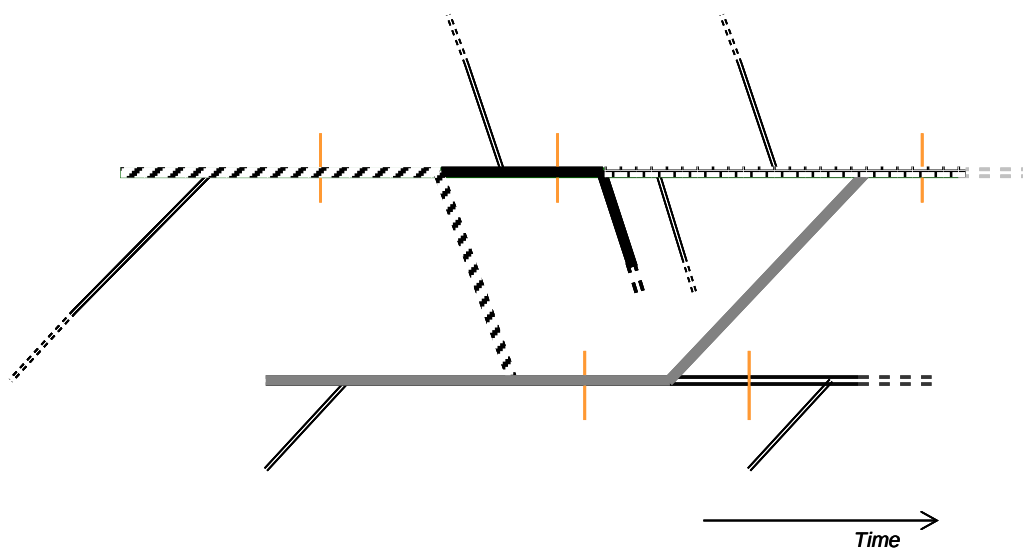


Fig.7a. Complex train coupling at Southern Railway



Continuous segments of reshaped train work are shown in textured lines

Fig.7b. Reshaped train work

Without the automatic reshaping function in TrainTRACS the planners previously tackled the problem manually mainly by juggling with the working schedule. Apart from being a tedious process, the manual schedule adjustments may also result in loss of efficiency. The pre-processing methods are beneficial to commuter train operators mostly because their train networks are complex and their services are intensive, for which extensive

train coupling is common. The problems are less relevant to inter-city train operations, for which train coupling is largely not practised in UK.

3.3 Incremental accumulation of domain knowledge

Real world scheduling is usually an evolving problem. For example, before privatisation in the early 1990s, the UK rail services were operated by one single national company British Rail. Since then, the UK rail service network has undergone several rounds of being partitioned, reformed and franchised to many TOCs; the train crew labour rules and practices have also been diverging between the privatised TOCs and they are still changing. Therefore it is essential for a train crew scheduling system to be able to keep up with the changing scheduling scenarios on a long term basis.

Another reason for the need to be able to incrementally enrich the scheduling system's domain knowledge model base is that it is impossible to acquire all such domain knowledge quickly in one go. Every TOC has its own peculiar problem scenarios to some extent and it is usually a very lengthy process spanning over many years in interacting with each TOC and discovering their precise problem scenarios.

It is therefore important in the long term that the scheduling system is built on an approach that is easily amenable to the incorporation of new domain knowledge from time to time. Otherwise the scheduling system would have to be overhauled frequently and that would be impractical in the long term.

Following a suitable approach, the need for further enhancements of the scheduling system would lessen over time as more and more problem scenarios would have been encountered, tackled and assimilated. Often, even if a new problem scenario has arisen, there would be some way, albeit probably a cumbersome one, of using the existing system to handle it. This is usually the first course of action to explore so that the train company users would not be held back, while a proper implementation would be incorporated into the system's research and development plan.

TrainTRACS follows the GaS approach discussed in section 2.1, which is advantageous because the scheduling rules, domain specific knowledge, can be dealt with almost entirely within the Generate phase. Furthermore, the Generate phase does not consider how the candidate shifts would fit amongst themselves, individual candidate shifts are generated independent of each other; application of the scheduling rules are therefore straightforward and the rule set can be expanded relatively easily – enabling TrainTRACS to engage in the process of generalisation and incorporation of new problem scenarios in a continuous incremental manner.

The current TrainTRACS has incrementally accumulated domain knowledge from a large number of transport operators over many years. That many TOCs have adopted TrainTRACS (see Section 1) lends support for the merit of its incremental approach. The wide range of problem scenarios that TrainTRACS has been used for is reflected in the variety of train operation types of the TOC users - Inter-city: NXEC, Virgin West Coast; Provincial: Northern, Cross Country, London Midland, Arriva Wales; Intensive Commuter Services: Northern, Southern, South Eastern; Urban city: London Overground.

4. Conclusions and remarks

Being a very hard optimization problem to solve, an automatic train crew scheduling system like TrainTRACS takes many years of incremental research and development to mature incorporating domain knowledge from the industry along the way. The GaS approach has shown to be particularly suitable.

Every TOC user of TrainTRACS has commissioned a pilot trial before they adopted the system. The trials were conducted using real data from the TOC involved. The TOCs would have verified, often by comparison with their existing schedules, that valid operable and efficient results have been produced before they accepted the system. Once adopted TrainTRACS is used in-house regularly for production use. Apart from LTP some TOCs are also using the system for STP, for which some new features and automatic links to other systems for other aspects of train operations planning have been developed in the TrainTRACS system.

The use of TrainTRACS has not replaced any train planners. On the contrary, train planners may have become more important because with much time freed up by using an automatic system, they can devote more effort on exploring many planning options and strategic issues like schedule robustness. Schedulers welcome the

use of TrainTRACS. Much stress is relieved from a job that is very much squashed in time between when the train unit diagrams are ready and the going live of the new set of timetables.

There may be fear that labour unions would strongly oppose computer produced schedules because efficiency savings may lead to job cuts. From the experience with TrainTRACS, this fear did not seem to have materialised. Many UK TOCs barely have sufficient train crews. Drivers in particular are expensive to be trained on specific routes before they are legally allowed to drive. At least initially, TOCs tend to turn possible efficiency savings in their crew schedules into making their operations more punctual and robust. In one case, the labour union seemed even more enthusiastic than the TOC management for TrainTRACS. The labour union wanted certain working patterns that management has agreed in principle, but the schedulers were unable to work out satisfactory schedules that can implement the requested working patterns. Trial application of TrainTRACS resulted in schedules that solved the problems.

References

- Abbink, E., Fischetti, M., Kroon, L., Timmer, G., & Vromans, M. (2005). Reinventing crew scheduling at Netherlands Railways. *Interfaces*, 35, 393-401.
- Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. Vance. (1998). "Branch-and-price: column generation for solving huge integer programs," *Opns. Res.* 46, 316-329.
- Burke, E., Pinedo, M., Steef van de Velde. (2001) Editorial, *J of Scheduling*, 4, 1-2.
- Caprara, A., M. Fischetti, P. Toth, D. Vigo, P. Luigi Guida. (1997). "Algorithms for railway crew management," *Mathematical Programming* 79, 125-141.
- Chvátal, V. (1979). "A greedy heuristic for the set-covering problem," *Mathematics of Operations Research* 4, 233-235.
- Desrochers, M. and F. Soumis. (1989). "A column generation approach to the urban transit crew scheduling problem," *Transportation Science* 23, 1-13.
- Fores, S., L. Proll, and A. Wren. (1998). "A column generation approach to bus driver scheduling," In: M. G. H. Bell (ed.) *Transportation Networks: Recent Methodological Advances*. Pergamon, 195-208.
- Fores, S., L. Proll, and A. Wren. (2002). "TRACS II: A hybrid IP/heuristic driver scheduling system for public transport," *JORS* 53, 1093-1100.
- Garey, M. R. and D. S. Johnson. (1979). "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, San Francisco.
- Huisman, D. (2007). "A column generation approach to solve the crew re-scheduling problem." *EJOR*, 180, 163-173.
- Hoffman, K. L., M. Padberg. (1993). "Solving Airline Crew Scheduling Problems by Branch-and-Cut," *Management Science* 39, No. 6, 657-682.
- Kroon, L.G., & Fischetti, M. (2001). "Crew Scheduling for Netherlands Railways "Destination: Customer"." In Voß, S., & Daduna, J.R. (eds), *Computer-Aided Scheduling of Public Transport*. Springer, Berlin, 181-201.
- Kwan, R. S. K. (2004). "Bus and train driver scheduling." In J. Y.-T. Leung (ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Chapter 51, 1- 20.
- Kwan, R. S. K. and A. Kwan. (2007). "Effective search space control for large and/or complex driver scheduling problems," *AOR* 155(1), 417-435.
- Laplagne, I. (2008). "Train driver scheduling with windows of relief opportunities," PhD Thesis, School of Computing, University of Leeds.
- Modern Railways. (2008) "West Coast dominates timetable changes." *Modern Railways*, December 2008, 46-47.
- Modern Railways. (2009) "West Coast Route Modernisation." *Modern Railways*, May 2009, 33-54.
- Smith, B. M. and A. Wren. (1988). "A bus crew scheduling system using a set covering formulation," *Transportation Research* 22A, 97-108.

- Willers, W. P., L. G. Proll, and A. Wren. (1995). "A dual strategy for solving the linear programming relaxation of a driver scheduling system," *AOR* 58, 519-531.
- Wren, A. (1981). "General review of the use of computers in scheduling buses and their crews." In A. Wren (ed.), *Computer scheduling of public transport*. North-Holland, 3-16.
- Wren, A. and J.-M. Rousseau. (1995). "Bus driver scheduling – an overview." In J. R. Daduna, I. Branco, and J. M. P. Paixao (eds.), *Computer-aided transit scheduling*. Springer-Verlag, 173-187.
- Wren, A., S. Fores, A. S. K. Kwan, R. S. K. Kwan, M. E. Parker, and L. Proll. (2003). "A flexible system for scheduling drivers," *J. of Scheduling* 6, 437-455.

Papers

On some combinatorial properties of PESP

Paolo Serafini

Abstract In this paper we investigate how fractional solutions of PESP can lead to discover violated inequalities by using some combinatorial properties based on sequencing ideas.

1 Introduction

After the first algorithms proposed for the Periodic Event Scheduling Problem [6], there has been some progress in speeding up the computation in order to cope with large size instances [1,3,5]. In this paper we try to understand some combinatorial properties of PESP, mainly based on sequence characterization, which may be helpful in reducing the computing time.

Let us recall the basic properties of PESP (Periodic Event Scheduling Problem). There are given:

- a directed graph $G = (N, E)$;
- a number $T \in \mathbb{R}^+$, called the period;
- intervals (also called spans) $D_e = [d_e^-, d_e^+]$, $e \in E$, with $d_e^-, d_e^+ \in \mathbb{R}$; ($d_e^- = d_e^+$ is allowed; w.l.o.g. $0 \leq d_e^+ - d_e^- < T$).

To each interval we associate the set in $\mathbb{R} \bmod T$ defined by (and by slight abuse of terminology also called interval or span)

$$\tilde{D}_e := \{ \tau \in \mathbb{R} \bmod T : (\tau - d_e^-) \bmod T \leq (d_e^+ - d_e^-) \bmod T \}$$

Each set $\tilde{D}_e$ is also represented by the set of ‘shifted’ spans

$$D_e(z) := [d_e^- - zT, d_e^+ - zT]$$

with $z \in \mathbb{Z}$. Let

$$\Delta_e := \bigcup_{z \in \mathbb{Z}} D_e(z)$$

Then $t \in \Delta_e \iff t \bmod T \in \tilde{D}_e$.

University of Udine, Dipartimento di Matematica e Informatica
E-mail: paolo.serafini@dimi.uniud.it

A schedule is a function $t : N \rightarrow R \bmod T$. A feasible schedule is such that $t \bmod T \in \tilde{D}_e$, $e \in E$. Alternatively a schedule is a function $t : N \rightarrow R$ and a feasible schedule is such that $t \in \Delta_e$, $e \in E$.

Then the PESP is: does there exist a feasible schedule?

PESP is NP-complete [6]. It remains NP-complete if the data are integers and consequently we may consider only integer-valued schedules. In this case it is NP-complete even if T is reduced to 3. Note that if t is feasible, also $t + c$ is feasible, with c a constant. Furthermore if t_i is feasible also $t_i + k_i T$ is feasible for any integer k_i .

In PESP nodes are associated to events, which in turn can correspond to starting or ending times of some task. Arcs correspond to constraints between events. Although these constraints look like ‘precedences’, the fact that the time model is circular makes the concept of ‘before’ and ‘after’ meaningless. These constraints model the fact that the time which has to elapse from one event to another one must be restricted to a specific time interval. The PESP is a useful model especially for finding periodic transportation time tables [1,2]. Hence it is important to improve the computational performance of related algorithms.

Fix $\hat{z}_e$, $e \in E$. Then finding t s.t. $t_j - t_i \in D_e(\hat{z})$, $(i, j) \in E$, is a polynomial problem solvable in time $O(n^3)$ (also called ‘Feasible Differential Problem’ [4]). Denote it as FDP($E, \hat{z}$). Let $\mathcal{C}$ be the set of all directed circuits in G . Given a circuit $C \in \mathcal{C}$ (a circuit has a definite orientation, the opposite circuit, denoted as $-C$, is another circuit in $\mathcal{C}$), a circuit span is defined as $[D^-(C), D^+(C)]$ where

$$D^+(C) := \sum_{e \in C^+} d_e^+ - \sum_{e \in C^-} d_e^-$$

$$D^-(C) := \sum_{e \in C^+} d_e^- - \sum_{e \in C^-} d_e^+$$

where C^+ is the set of arcs directed as the circuit and C^- is the set of arcs directed opposite to the circuit. Note that $D^+(-C) = -D^-(C)$ and $D^-(-C) = -D^+(C)$. Let us associate to each arc $e = (i, j)$ two distances

$$\begin{aligned} d_{ij} &:= d_e^+ - \hat{z}_e T \\ d_{ji} &:= -d_e^- + \hat{z}_e T \end{aligned} \tag{1}$$

Then the length of a circuit C with respect to (1) (by computing for each arc the distance according to the orientation) is given by $D^+(C) - \hat{z}(C)T$ where $\hat{z}(C) := \sum_{e \in C^+} \hat{z}_e - \sum_{e \in C^-} \hat{z}_e$.

A necessary and sufficient condition for the feasibility of the FDP($E, \hat{z}$) is given by

$$0 \in [D^-(C) - \hat{z}(C)T, D^+(C) - \hat{z}(C)T] \quad C \in \mathcal{C}$$

i.e.

$$D^+(C) - \hat{z}(C)T \geq 0 \quad C \in \mathcal{C}$$

This immediately implies that PESP is feasible if and only if there exist integer numbers z_e satisfying the set of inequalities:

$$D^+(C) - z(C)T \geq 0 \quad C \in \mathcal{C} \tag{2}$$

and denoting

$$\delta(C) := \left\lfloor \frac{D^+(C)}{T} \right\rfloor$$

we may also say that PESP is feasible if and only if there exist feasible integers z_e satisfying

$$z(C) \leq \delta(C) \quad C \in \mathcal{C} \quad (3)$$

Note that if (3) is feasible then there exist infinite feasible solutions given by a generic solution of (3) plus any solution in the kernel of the arc-circuit incidence matrix of the graph. It is easily shown that we may freely set $z_e = 0$ on all arcs of any spanning tree S . Then (3) is equivalent to

$$\begin{aligned} z(C) &\leq \delta(C) & C \in \mathcal{C} \\ z_e &= 0 & e \in S \end{aligned} \quad (4)$$

2 Event sequencing

Let us first note that we may assume w.l.o.g. that $0 < d_e^- \leq d_e^+ < T$ and $d_e^+ - d_e^- \leq T - 2$ for any $e \in E$. Indeed, with integral data the span $[d_e^-, d_e^+]$ is redundant if $d_e^+ - d_e^- \geq T - 1$ and can be dropped together with the corresponding arc. As for the other remark if $d_e^- \leq 0 \leq d_e^+$ for a certain arc e , it is always possible to replace the arc $e = (i, j)$ with two arcs in series (i, k) and (k, j) with spans

$$\begin{aligned} [d_{ik}^-, d_{ik}^+] &:= [\lfloor (T + d_e^-)/2 \rfloor, \lfloor (T + d_e^+)/2 \rfloor] \\ [d_{kj}^-, d_{kj}^+] &:= [\lceil (T + d_e^-)/2 \rceil, \lceil (T + d_e^+)/2 \rceil] \end{aligned}$$

Note that the new node k is only linked to i and j . Since $T + d_e^- \geq 2$ (implied by $d_e^+ \geq 0$ and $d_e^+ - d_e^- \leq T - 2$), it follows $\lfloor (T + d_e^-)/2 \rfloor > 0$. Similarly, since $T + d_e^+ \leq 2T - 2$ (implied by $d_e^- \leq 0$ and $d_e^+ - d_e^- \leq T - 2$) it follows $\lceil (T + d_e^+)/2 \rceil < T$.

Definition 1 We say that the constraints are *regular* if $0 < d_e^- \leq d_e^+ < T$ holds for all arcs.

Now we want to show that the equation $z(C) = q$ has a precise combinatorial meaning under the assumption of regularity.

Definition 2 A circuit C for which the constraint $z(C) = k - |C^-|$ is imposed is said to be *k-conditioned* and this constraint is called *k-condition*.

In the sequel, whenever we say ‘feasible’ we mean feasible with respect to the constraints (3) and any other possible *k-condition* explicitly mentioned.

Definition 3 An ordered subset of events $i_1, i_2, \dots, i_h$ is *sequenced* if there exists a feasible schedule such that $t_1 < t_2 < \dots < t_h < t_1 + T$.

Note that the ordered set $i_2, i_3, \dots, i_h, i_1$ is sequenced as well. Indeed the property refers to a circular ordering of the events. Then we have

Proposition 1 *The events in a circuit C , ordered as C , are sequenced if and only if C is 1-conditioned and there exists a corresponding feasible integer z .*

Proof Let $t_1, t_2, \dots, t_h$ ($h = |C|$) be a schedule such that the respective events are sequenced. By definition of sequencing we have

$$z_e = \begin{cases} 0 & \text{if } e \in C^+ \text{ and } e \neq (h, 1), \\ 1 & \text{if } e \in C^+ \text{ and } e = (h, 1), \\ -1 & \text{if } e \in C^- \text{ and } e \neq (h, 1), \\ 0 & \text{if } e \in C^- \text{ and } e = (h, 1), \end{cases}$$

whence $z(C) = 1 - |C^-|$.

Conversely let us suppose t is a feasible schedule and the constraint $z(C) = 1 - |C^-|$ is satisfied. Let us first assume that $C^- = \emptyset$. Now let $\tau_i = t_i \bmod T$ and $\nu_e = \tau_j - \tau_i$, $e = (i, j)$. As already observed τ is a feasible schedule as well. Let z'_e the derived values so that $\nu_e \in D_e(z'_e)$. Then $z'(C) = z(C)$, for any C . Since $z'_e \in \{0, 1\}$, there must be exactly one arc such that $\nu_e \in D_e(1)$. We call this arc *switching* arc. For the other $h-1$ arcs $\nu_e \in D_e(0)$. Now let us order the arcs starting from the arc next to the switching one and following the orientation of the circuit. The schedule τ satisfies the definition of sequencing. If $C^- \neq \emptyset$ the constraint $-d_e^+ + z'_e T \leq \tau_i - \tau_j \leq -d_e^- + z'_e T$ if $e \in C^-$, implies that the values of z'_e are decreased by -1 with respect to direct orientation, whence the thesis. $\blacksquare$

Note that $z(C) \geq 1 - |C^-|$ for any feasible schedule and that this lower bound is attained only if the events are sequenced. Otherwise $z(C) > 1 - |C^-|$. Let us consider the reverse circuit $-C$. Then from the previous results $z(-C) \geq 1 - |-C^-|$. Since $z(-C) = -z(C)$ and $|-C^-| = |C^+|$, we have $z(C) \leq |C^+| - 1$. It is interesting to compare this bound with (4). Under the regularity assumption, $\delta(C) \leq |C^+| - 1$ so that (4) implies the new bound.

We have therefore shown that $z(C) = 1 - |C^-|$ imposes sequencing the events in C and $z(C) = |C^+| - 1$ imposes the reverse sequence in C . What about the intermediate values? To this aim we say:

Definition 4 An ordered subset of events $i_1, i_2, \dots, i_h$ is *k-sequenced* if there exists a feasible schedule such that $t_1 < t_2 < \dots < t_h < t_1 + kT$, and $t_{i+1} - t_i < T$, $i = 1, \dots, h-1$, $t_1 + kT - t_h < T$.

A first property about *k*-sequencing is given by:

Proposition 2 An ordered subset of events $i_1, \dots, i_h$ is *k-sequenced* if and only if the reverse set $i_h, \dots, i_1$ is $(h-k)$ -sequenced.

Proof By definition of *k*-sequencing we have

$$\begin{aligned} 0 &< t_{i+1} - t_i < T & i = 1, \dots, h-1 \\ (k-1)T &< t_h - t_1 < kT \end{aligned}$$

The schedule $t'_i = t_i + (h-i)T$ is feasible as well. It is immediate to verify that

$$\begin{aligned} 0 &< t'_i - t'_{i+1} < T & i = 1, \dots, h-1 \\ (h-k-1)T &< t'_1 - t'_h < (h-k)T \end{aligned}$$

$\blacksquare$

The generalization of Proposition 1 is given by:

Proposition 3 *The events in C , ordered as C , are k -sequenced if and only if C is k -conditioned and there exists a corresponding feasible integer z .*

Proof We parallel the proof of Proposition 1, supposing first that $C^- = \emptyset$. Now there must be k arcs such that $\nu_e \in D_e(1)$, whereas the other $h - k$ arcs satisfy $\nu_{ae} \in D_e(0)$. Call these k arcs switching arcs. Now let us order the arcs starting from any arc next to a switching one and following the orientation of the circuit. With respect to this new order consider the schedule t'' obtained from τ by adding one more T each time a switching arc is reached. The schedule t'' satisfies the definition of k -sequencing. The ‘only if’ part of the proof is trivial. ■

In our terminology when we say that the events of a circuit C are k -conditioned, we mean that only the condition $z(C) = k - |C^-|$ has been imposed, and nothing is said about the actual possibility of k -sequencing the circuit, because other constraints may prevent it. We say that the events are k -sequenced if indeed there exists a feasible schedule fulfilling the k -sequence condition.

3 Compatibility among sequences

The system (4) leads to the definition of the following polyhedron:

$$P := \left\{ x \in R^{|E|-|N|+1} : x(\tilde{C}) \leq \delta(C), \quad C \in \mathcal{C}(E) \right\}$$

where $\tilde{C} = C \cap (E \setminus S)$. P contains the polyhedron P_z convex hull of all integer feasible solutions of (4). Let us consider also the following polyhedron:

$$P_0 := \left\{ x \in R^{|E|-|N|+1} : x(\tilde{C}) \leq D^+(C)/T, \quad C \in \mathcal{C}(E) \right\}$$

A feasible solution of P_0 can be found in polynomial time since the separation problem can be easily solved by any algorithm detecting negative length circuits. It is not difficult to show that P_0 is never empty. Furthermore P_0 and P share the same set of integral solutions.

The separation problem for P is difficult in general. If x is integral then the membership and separation problems for P and P_0 are the same and according to the previous observation can be carried out in polynomial time. We want to investigate the membership and separation problem in case x is fractional by using sequencing properties.

The aim of the computation is to find an integer solution in P or to establish that there is none. Usually the algorithms start with known solutions for a subset of circuits and they try to extend this solution to the whole set of circuits, by following a definite scheme of circuit insertion.

Here we think of starting with an initial set of circuits and find a feasible vertex solution for this set. The vertex can be either integer or fractional. If it is integer checking its feasibility for P is easy and can either generate a new circuit to add to the set or stop the computation. If it is fractional, the constraints

$$x(\tilde{C}) = \delta(C), \quad C \in \bar{\mathcal{C}} \tag{5}$$

defining the vertex, bring an inconsistency among the circuit sequences. More exactly each circuit C is $(\delta(C) + |C^-|)$ -conditioned (or equivalently each circuit $-C$ is $(|C^+| - \delta(C))$ -conditioned). If these conditions are incompatible there can be no integer x feasible for (4) and (5).

An interesting question is whether sequencing properties may lead to discover violated inequalities in (4) for a fractional x . We have the following results:

Definition 5 Two circuits are incompatible if there exists no sequence of all nodes containing both circuits. ■

Lemma 1 Two circuits are incompatible if and only if there exist 3 nodes in both circuits, say i, j and k which are traversed as $i \rightarrow j \rightarrow k$ in one circuit and as $i \rightarrow k \rightarrow j$ in the other circuit.

Proof Sufficiency is obvious. For the necessity let us suppose that any common triple of nodes is sequenced in the same way in both circuits. Starting from a triple, say i, j and k and considering in turn each common node with respect to other two common nodes, we see that the subset of common nodes must be sequenced in the same way in both circuits. Then we may easily form a sequence of all nodes such that the two circuits are contained as subsequences. ■

The next lemma shows that no vertex of P can be associated with incompatible 1-conditioned circuits.

Lemma 2 Let C_1 and C_2 be two elementary circuits with 3 nodes in common, say i, j and k . Suppose $C_1 = i \rightarrow j \rightarrow k \rightarrow i$ and $C_2 = i \rightarrow k \rightarrow j \rightarrow i$. Then the equations $x(C_1) = |C_1^+| - 1$ and $x(C_2) = |C_2^+| - 1$ cannot hold at the same time if $x \in P$.

Proof By patching together the path $i \rightarrow j$ from C_1 and $j \rightarrow i$ from C_2 we form a new circuit. Similarly we form two other circuits by considering the pairs i, k and j, k . Starting from these three circuits we possibly generate new circuits from a non-elementary circuit (no repeated arcs) until all generated circuits are elementary. Let s be the number of currently generated circuits, and t be the number of arc cancellations which occur in the generation of new circuits. Initially $s = 3$ and $t = 0$. Suppose the circuit $i \rightarrow j \rightarrow i$ is not elementary. So there is an arc $p \rightarrow q$ which is traversed twice. We have two cases: either $i \rightarrow p \rightarrow q \rightarrow j \rightarrow p \rightarrow q \rightarrow i$ or $i \rightarrow p \rightarrow q \rightarrow j \rightarrow q \rightarrow p \rightarrow i$. In the former case we generated the two circuits $i \rightarrow p \rightarrow q \rightarrow i$ and $j \rightarrow p \rightarrow q \rightarrow j$ from a single circuit so s is increased by 1 and t is unchanged since there are no cancellations. In the latter case the arcs $p \rightarrow q$ is cancelled since it is traversed in opposite directions and we generate the two circuits $i \rightarrow p \rightarrow i$ and $j \rightarrow q \rightarrow j$. However it may happen that either circuit (or perhaps both) collapse into a node if $i = p$ and/or $j = q$. In any case the number $s + t$ is not decreased. Eventually we have a set of circuits $\hat{C}_i, i = 1, \dots, s$.

By assuming $x \in P$, $x(\hat{C}_i) \leq \delta(C_i) \leq |\hat{C}_i^+| - 1, i = 1, \dots, s$. By summing $\sum_i x(\hat{C}_i) \leq \sum_i |\hat{C}_i^+| - s$. Now note that $x(C_1) + x(C_2) = \sum_i x(\hat{C}_i)$ and that $\sum_i |\hat{C}_i^+| = |C_1^+| + |C_2^+| - t$ (each arc cancellation lowers by one either $|C_1^+|$ or $|C_2^+|$ but not both). Then we have

$$x(C_1) + x(C_2) \leq |C_1^+| + |C_2^+| - s - t \leq |C_1^+| + |C_2^+| - 3$$

whence the thesis. ■

Definition 6 Given a set $\bar{C} = \{C_1, C_2, \dots, C_p\}$ of circuits, any set of circuits obtained as a circuit partition of the arcs of $\bar{C}$, after possible cancellations of pairs of oppositely directed arcs, is called *generated* from $\bar{C}$.

Note that a pair of oppositely directed arcs can be viewed to as a trivial circuit of two nodes, for which 1-condition always holds, so that a cancellation can be regarded as a trivial circuit.

Lemma 3 Given a set $\bar{C}$ of p 1-conditioned circuits and a point $x \in P$ (not necessarily integer) feasible w.r.t. them, any set $\hat{C}$ of q circuits, generated from $\bar{C}$ with t cancellations, is formed by 1-conditioned circuits if and only if $p = q + t$.

Proof $\sum_{C \in \bar{C}} x(C) = p - \sum_{C \in \bar{C}} |C^-|$. Also $\sum_{C \in \bar{C}} x(C) = \sum_{C \in \hat{C}} x(C)$, and $\sum_{C \in \bar{C}} |C^-| = \sum_{C \in \hat{C}} |C^-| + t$. Hence $\sum_{C \in \hat{C}} x(C) = p - t - \sum_{C \in \hat{C}} |C^-|$. Since $x(C) \geq 1 - |C^-|$ holds for any circuit by assumption, then $\sum_{C \in \hat{C}} x(C) \geq q - \sum_{C \in \hat{C}} |C^-|$. By comparing with the previous equality the thesis follows. ■

Note that the generated set of circuits can be such that $q + t < p$, even with a feasible x . In this case at least one of the generated circuits is k -sequenced with $k > 1$.

The opposite case, $p < q + t$ is quite interesting as the following lemma shows.

Lemma 4 Given a set $\bar{C}$ of p 1-conditioned circuits, and a (not necessarily integer) $x \in P$ feasible w.r.t. these constraints, if there exists a set $\hat{C}$ of q circuits, generated from $\bar{C}$ with t cancellations, and $p < q + t$, then x violates at least one of the inequalities $x(C) \geq 1 - |C^-|$, $C \in \hat{C}$.

Proof The thesis can be derived by following the same line of reasoning of the previous lemma. ■

We may extend the previous result as:

Lemma 5 Given a set $\bar{C}$ of circuits, with $p(\bar{C}) := \sum_{C \in \bar{C}} |C^+| - \delta(C)$, and a (not necessarily integer) $x \in P$ feasible for $x(C) = \delta(C)$, all $C \in \bar{C}$, if there exists a set $\hat{C}$ of circuits, generated from $\bar{C}$ with t cancellations, and $p(\bar{C}) < p(\hat{C}) + t$, then x violates at least one of the inequalities $x(C) \leq \delta(C)$, $C \in \hat{C}$.

Note that if x is infeasible, the fact that $p(\bar{C}) = p(\hat{C}) + t$ does not imply the corresponding conditioning of the reversed generated circuits (as in Lemma 3). Indeed it may happen that one of the generated circuits gives rise to a violated inequality, so that other circuits are not necessarily correctly conditioned. However, since we are primarily interested in finding violated inequalities if x is not feasible, we just have to check each inequality of the generated set. If they are not violated, then they are all correctly conditioned.

4 Example

Consider the instance shown in Figure 1(a). The spanning tree and the numbering of the chords is shown in Figure 1(b). $T = 10$.

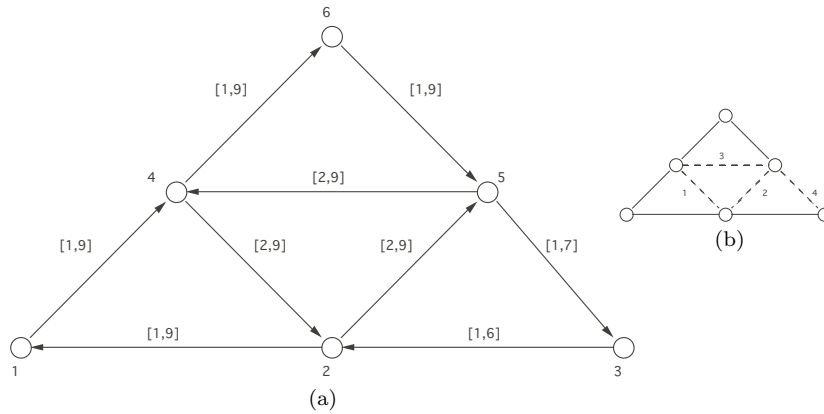


Fig. 1

Consider the four circuits $C_1 = (4, 2, 5, 6, 4)$, $C_2 = (1, 2, 5, 4, 1)$, $C_3 = (5, 3, 2, 4, 5)$ and $C_4 = (3, 5, 6, 4, 1, 2, 3)$ ($\bar{C} = \{C_1, C_2, C_3, C_4\}$) giving raise to the following system associated with a vertex

$$\begin{aligned} x_1 + x_2 &= 1 \\ x_2 + x_3 &= 1 \\ -x_1 - x_3 + x_4 &= 0 \\ -x_4 &= -1 \end{aligned} \quad (6)$$

for which $\hat{x}_1 = \hat{x}_2 = \hat{x}_3 = 1/2$, $\hat{x}_4 = 1$. We get $p(\bar{C}) = 5$. The set $\bar{C}$ generates the circuit $C_5 = (1, 2, 5, 6, 4, 1)$ repeated twice (i.e. $\hat{C} = \{C_5, C_5\}$) with 4 cancellations. We get $\delta(C_5) = 0$, $C_5^+ = 1$ whence $p(\hat{C}) = 2$. Therefore $p(\bar{C}) < p(\hat{C}) + t$, and C_5 corresponds to a violated inequality. Indeed $\hat{x}(C_5) = \hat{x}_2 = 1/2 > \delta(C_5) = 0$. Adding this inequality we get an integer solution $z_1 = 1$, $z_2 = 0$, $z_3 = 1$, $z_4 = 2$ which does not violate any circuit inequality and therefore it corresponds to the feasible schedule $t_1 = 5$, $t_2 = -1 (= 9)$, $t_3 = -5 (= 5)$, $t_4 = 6$, $t_5 = 8$, $t_6 = 7$.

Alternatively, we may reason as follows. The equation $x(C_3) = 0$ imposes the 2-sequence $(5, 4, 2, 3)$ (since $|C_3^+| = 2$), $x(C_4) = 1$ imposes the sequence $(3, 2, 1, 4, 6, 5)$. Note that the elements of the former sequence are contained in the latter. The elements of the second sequence are sequenced over one period, so that we impose the sequence $(3, 2, 1, 4, 6, 5, 3, 2, 1, 4, 6, 5)$ over two periods. It is clear impossible to sequence over two periods the sequence $(5, 4, 2, 3)$. Then the two equations cannot hold simultaneously.

Since we always have $x(C_3) \leq 0$, and $x(C_4) \leq -1$, by summing the inequalities we derive $x(C_3) + x(C_4) \leq -1$ i.e. $-x_1 - x_2 \leq -1$. However, since this inequality cannot hold as equality, because of the contradictory sequencing requirements, we may derive the stricter inequality $-x_1 - x_2 \leq -2$, which cuts the previous vertex.

We could also note that the set $\{C_3, C_4\}$ produces $p(\{C_3, C_4\}) = 3$ and generates the circuits $C_6 = (1, 2, 4, 1)$ and $C_7 = (4, 5, 6, 4)$ with 2 cancellations. Since $p(\{C_6, C_7\}) = 2$ we get either C_6 or C_7 (or both) corresponds to a violated inequality. Indeed $\hat{x}(C_6) = -1/2 > \delta(C_6) = -1$ and $\hat{x}(C_7) = -1/2 > \delta(C_7) = -1$.

References

1. Liebchen C., R. H. Möhring: The modeling power of the periodic event scheduling problem: Railway timetables - and beyond. In: F. Geraets, L. Kroon, A. Schbel, D. Wagner, and C. D. Zaroliagis (eds), *Algorithmic Methods for Railway Optimization*, vol. 4359 of Lecture Notes in Computer Science, Springer, Berlin/Heidelberg, pp. 3-40 (2007).
2. Lindner T.: Train schedule optimization in public rail transport. Tesi di dottorato, Fachbereich für Mathematik und Informatik der Technischen Universität Braunschweig (2000).
3. Odijk M.A.: A constraint generation algorithm for the construction of periodic railway time-tables, *Transportation Research B*, **30**, (1996).
4. Rockafellar R.T.: *Network flows and monotropic optimization*. John Wiley & Sons, New York, NY, USA (1984).
5. Schrijver A., A. Steenbeek: Dienstregelingontwikkeling voor Railned (Timetable construction for Railned). Technical Report, Centrum voor Wiskunde en Informatica, Amsterdam (1994).
6. Serafini P., W. Ukovich: A mathematical model for periodic scheduling problems, *SIAM Journal on Discrete Mathematics*, **2**, 550-581 (1989).

Polynomial algorithm for $1 \mid r_j, p_j = p, pmnt \mid \sum w_i c_j$ scheduling problem

Alexander Kvaratskhelia · Alexander Lazarev

Abstract In this paper, we consider the minimizing total weighted completion time in preemptive equal-length job with release dates scheduling problem on a single machine. Before this paper the problem is known to be open. Here, we present a polynomial time algorithm that solves the problem with $O(n^7)$ operations.

1 Introduction

The *minimizing total weighted completion time in preemptive equal-length job with release dates* scheduling problem on a single machine is formulated as follows.

There are n jobs that have to be processed on a single machine with preemptions. Let $N = \{1, \dots, n\}$ be a set of jobs. For each jobs $j \in N$ the following parameters are given: release date $r_j \geq 0$, identical processing time $p_j = p > 0$, and weight $w_j \geq 0$. All parameters are integer values. Without loss of generality we suppose that $w_1 \geq \dots \geq w_n$ and $\min_{j \in N} r_j = 0$.

A schedule $\mathbb{S}$ sets an order in which jobs are executed on the machine. Basically, we consider a schedule as a piecewise constant left-continuous function $\mathbb{S}(t)$ that is acting from $\mathbb{R}$ to $\{0, \dots, n\}$, i.e. $\mathbb{S} : \mathbb{R} \rightarrow N \cup \{0\}$. If $\mathbb{S}(t)$ equals $j \in N$ then we say that job j is executed on the machine. Otherwise (when $\mathbb{S}(t) = 0$), we say the machine is idle. As all parameters of jobs are integer values, at the rest of the paper we consider only integer time point t . A part of a job with length 1 we call as *unit*.

For a given schedule, each job j has *start time* S_j and *completion time* C_j when execution of this job finishes. The completion time of job j is defined as the largest integer t such that $\mathbb{S}(t) = j$, the start time is defined as the smallest integer t such that $\mathbb{S}(t + 1) = j$.

Alexander Kvaratskhelia
Institute of Control Science of the Russian Academy of Science
E-mail: akvarats@gmail.com

Alexander A. Lazarev
Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya st. 65, 117997, Moscow, Russia. The work is part of the project supported by the program of Presidium of the Russian Academy of Sciences N 29 "The mathematical theory of control".
E-mail: jobmath@mail.ru

Let $F(\mathbb{S})$ denote the value of *total weighted completion time* $\sum_{j=1}^n w_j C_j$ for schedule π . The objective is to find an optimal schedule $\mathbb{S}^*$ that minimizes *total weighted completion time*. In the notation of Graham et al. [1] the problem is denoted as

$$1 \mid r_j, p_j = p, pmnt \mid \sum w_i C_j.$$

There exists a polynomial time algorithm for non-preemptive case of the problem [2] with complexity $O(n^7)$ operations.

The rest of this paper is organized as follows. In Section 2 we give some properties of optimal schedules. In Section 3 we show that optimal solution can be found among permutation schedules. The polynomial algorithm is presented in Section 4.

2 Properties of optimal schedules

In this section we present some properties of optimal schedules for the problem.

Theorem 1 *For each optimal schedule we have: if $S_i < S_j$ then $C_i > C_j$ for all $i, j \in N$.*

Proof Consider an optimal schedule $\mathbb{S}$ for which we have $S_i < S_j$ and $C_i < C_j$ (see Fig. 1).

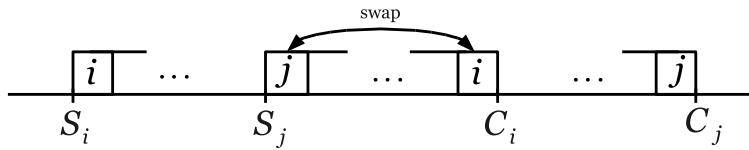


Figure 1.

Here, we can swap the first item of job j with the last item of job i and improve $\mathbb{S}$ by value w_i . This contradiction proves the theorem. $\triangleleft$

Let $\mathbb{I}_j$ denote a set of time points when job j is interrupted in a schedule $\mathbb{S}$, i.e. for each $t \in \mathbb{I}_j$ we have $\mathbb{S}(t) = j$, $\mathbb{S}(t+1) \neq j$, and $t \neq C_j$.

Denote by $\mathbb{I} = \bigcup_{j \in N} \mathbb{I}_j$ and $\mathbb{T}_k$ the interval $(r_k, r_{k+1}]$, $k = 1, \dots, n$, where $r_{n+1} = np$.

Lemma 1 *For each optimal schedule we have:*

$$\forall j \in N, \forall k \in N \quad |\mathbb{I}_j \cap \mathbb{T}_k| \leq 1,$$

i.e., each job is interrupted no more than once on $\mathbb{T}_k$.

Proof Consider an optimal schedule $\mathbb{S}$ where for some j and k we have $|\mathbb{I}_j \cap \mathbb{T}_k| > 1$, i.e. job j is interrupted at least twice on $\mathbb{T}_k$ (see Fig. 2).

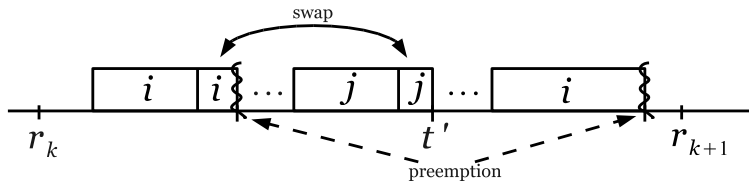


Figure 2.

On this figure job j is finished at time t' . Otherwise, by Theorem 1, schedule $\mathbb{S}$ is not optimal one. In this case, we can swap units of jobs as shown on the figure and improve $\mathbb{S}$ at least by value w_j . This contradiction proves the lemma. $\triangleleft$

Lemma 2 *In each optimal schedule all interrupted jobs are processed at the end of $\mathbb{T}_k$, $\forall k \in N$.*

Proof Consider an optimal schedule $\mathbb{S}$ where for some $k \in N$ there exist two jobs:

1. job j that is interrupted on $\mathbb{T}_k$;
2. job i starts immediately after j was interrupted and completed either on $\mathbb{T}_k$, or somewhere afterward.

The point is that both the jobs are available on $\mathbb{T}_k$ and, from Theorem 1, job i is finished before job j . Therefore, we can swap parts of these jobs in such way that interrupted part of j will be processed after i is completed. Moving job j at right does not change the cost function of the schedule, whereas moving of job i at left strongly improve the schedule.

Lemma 3 *For each optimal schedule, we have:*

$$|\mathbb{I} \cap \mathbb{T}_k| \leq 1, \text{ for each } \mathbb{T}_k$$

i.e., at most one job can be interrupted on $\mathbb{T}_k$.

Proof Consider a schedule π where we have the following situation:

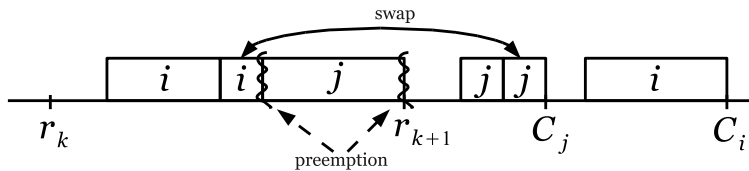


Figure 3.

In this case, we can swap units of jobs as shown on the figure and strongly improve the schedule by value w_j . $\triangleleft$

Theorem 2 *For each optimal schedule we have:*

1. $\mathbb{I} \subseteq \{r_2, r_3, \dots, r_n\}$;
2. if some job i is interrupted at r_k then $S_k = r_k$.

So in each optimal schedule, jobs can be interrupted only when another job becomes available. Moreover, job can be interrupted at r_k only by job k .

Proof The first part of the theorem directly follows from Lemmas 1–3. Now we show that if some job i is interrupted at r_k then $S_k = r_k$.

Consider an optimal schedule $\mathbb{S}$ where job i is interrupted at r_k but then job j starts, $r_j < r_k$. Due to Theorem 1, we have $C_j < C_i$ (see Fig. 4).

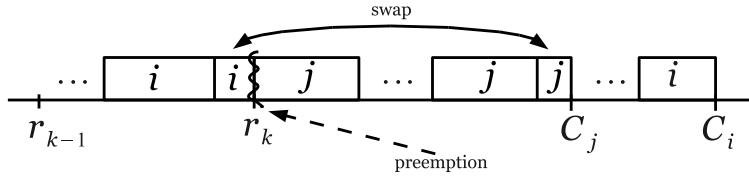


Figure 4.

Since $r_j < r_k$, we can swap items of jobs i and j as shown on the figure, and improve the schedule by value w_j . The theorem is proved. $\triangleleft$

Theorem 3 For each optimal schedule we have: if $r_i \leq r_j$ and $w_i > w_j$ then $C_i < C_j$.

Proof Consider a schedule π in which we have:

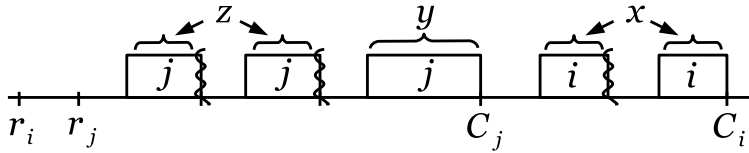


Figure 5.

Here, x denotes the length of job's i parts that are processed after C_j ; y denotes the length of job's j last part; and z denotes value $p - y$. Note, z equals zero if job j is never interrupted in π . Let $\Delta C = C_i - C_j$.

If $x \leq y$ we construct the following schedule π' :

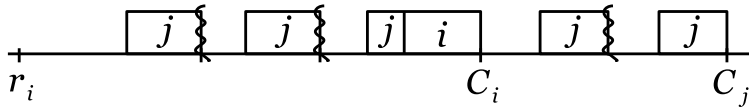


Figure 6.

In this case, we have $F(\pi') - F(\pi) = (w_j - w_i)\Delta C < 0$.

If $x > y$ we construct the following schedule π' :

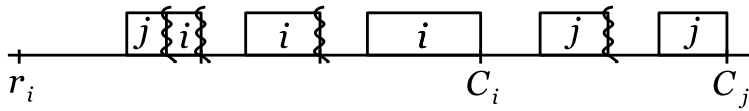


Figure 7.

Note that $x \leq y + z$, because $r_i \leq r_j$ and the overall processing times of both the jobs are the same (equal p). In this case, we also have $F(\pi') - F(\pi) = (w_j - w_i)\Delta C < 0$.
◁

Corollary 1 *For each optimal schedule we have: if $r_i \leq S_j$ and $w_i > w_j$ then $C_i < C_j$.*

Proof In the proof of Theorem 3, the point is that we can move job i at left and strongly improve schedule $\mathbb{S}$. This movement can be done not only in case $r_i \leq r_j$, but also in case $r_i \leq S_j$. Therefore, we can repeat our reasoning for $r_i \leq S_j$ and prove this corollary.
◁

3 Permutation schedules

As we mentioned in Introduction, the general way to define a schedule is to use a function acting from $\mathbb{R}$ to $\{0, 1, \dots, n\}$. For a variety of scheduling problems (typically, without preemptions) one can consider a schedule as a permutation of job's indices. In this section we show that in spite of preemptions we can find an optimal solution of the problem in permutation schedules.

By π we denote a permutation schedule, i.e. $\pi = (j_1, j_2, \dots, j_n)$ where $j_k \in N$. We say that job's order $(j_1, j_2, \dots, j_n)$ means $C_{j_1} < C_{j_2} < \dots < C_{j_n}$.

Now we give a procedure that constructs a time schedule $\mathbb{S}$ from a permutation schedule π .

Procedure 1 (*construct time schedule from permutation*)

- 1: jobs j_1 starts at r_{j_1} and ends at $r_{j_1} + p$ (i.e. $S_{j_1} := r_{j_1}$, $C_{j_1} := r_{j_1} + p$);
- 2: mark interval $[r_{j_1}, r_{j_1} + p)$ as busy;
- 3: **for** $k = 2, 3, \dots, n$ **do**
- 4: **if** r_{j_k} is marked **then**
- 5: let s be the nearest value that is unmarked and greater than r_{j_k} ;
- 6: **else**
- 7: $s := r_{j_k}$;
- 8: **end if**
- 9: calculate width of unmarked intervals between s and $C_{j_{k-1}} + 1$. Let this value be d (d can also be negative);
- 10: **if** $d \leq p$ **then**
- 11: job j_k starts at s and fills unmarked intervals to the right until is finished;
- 12: **else**
- 13: job j_k ends at $C_{j_{k-1}} + 1$ and fills unmarked intervals to the left;
- 14: **end if**
- 15: **end for**

On each step of the procedure we need to perform the following operations:

- realize r_{j_k} is in unmarked interval or not. This can be done by binary search among start and end times of jobs in $O(\log n)$ operations;
- calculate the width of unmarked intervals between r_{j_k} and $C_{j_{k-1}} + 1$. This can be done in $O(n)$ operations.

Therefore, each step is executed in $O(n)$ operations. Since Procedure 1 has n steps, the overall complexity of the procedure is $O(n^2)$.

Consider an optimal schedule $\mathbb{S}$ with completion times $C_1, \dots, C_n$ respective permutation $\pi = (j_1, j_2, \dots, j_n)$. Let apply Procedure 1 to this permutation and obtain schedule $\mathbb{S}'$ with completion times $C'_1, \dots, C'_n$.

Theorem 4 *For each optimal schedule $\mathbb{S}$ we have $C_j = C'_j$, $j \in N$.*

Proof In schedule $\mathbb{S}'$, job j_1 is completed at the least possible time $r_{j_1} + p$. If in optimal schedule $\mathbb{S}$ we have $C_{j_1} > C'_{j_1}$ then there exists a unit of some job $i \neq j_1$ that is processed before C_{j_1} . Since j_1 is the first completed job, we can apply Theorem 1 and show that schedule $\mathbb{S}$ is not optimal one. Therefore, $C_{j_1} = C'_{j_1}$.

Suppose that for $i < k$ the equalities $C_{j_i} = C'_{j_i}$ hold. Now, we show $C_{j_k} = C'_{j_k}$. Note that job j_k cannot be finished earlier than C'_{j_k} in $\mathbb{S}$ because, due to Procedure 1, either restriction $C_{j_k} \geq r_{j_k} + p$, or $C_{j_k} > C_{j_{k-1}}$ will be violated.

If $C_{j_k} > C'_{j_k}$ then there exists a unit of some job i from set $\{j_{k+1}, \dots, j_n\}$ that is processed after S_{j_k} and before C_{j_k} . In this case we can also apply Theorem 1 and prove schedule $\mathbb{S}$ is not optimal one. This contradiction proves $C_{j_k} = C'_{j_k}$.

The theorem is proved. $\triangleleft$

Theorem 4 states that we can find an optimal solution among permutation schedules. We use this property in our algorithm.

Lemma 4 *If permutation π conforms to the optimal time schedule then line 13 of Procedure 1 is never executed.*

Proof Indeed, line 13 means that between possible start time s of the current job j_k and the minimal completion time $C_{j_{k-1}} + 1$ there are units of job j_k and some other job i (that is completed after j_k). In this case, we can always exchange the last unit of j_k with this unit of job i and strongly improve the schedule by value w_{j_k} . $\triangleleft$

Theorem 5 *In every optimal schedule, each part of each job starts and ends at time points from set $T = \{r_j + lp, j = \overline{1, n}, l = \overline{0, n}\}$.*

Proof Taking into account Lemma 4, this theorem directly follows from the structure of time schedule that is constructed by Procedure 1. $\triangleleft$

Note, that the number of elements in set T is $O(n^4)$.

Consider a permutation $\pi = (j_1, j_2, \dots, j_n)$. We say that jobs $j_i, j_{i+1}, \dots, j_m$ organize a group if:

- execution of jobs in group starts and ends with job j_m ;
- all jobs $j_1, \dots, j_{i-1}$ are executed completely before jobs from the group;
- all jobs $j_{m+1}, \dots, j_n$ are executed completely after jobs from the group.

Here is an example of a group:

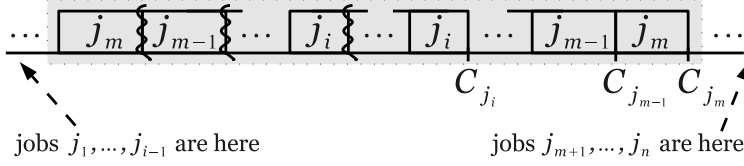


Figure 8.

To denote a group we use symbol G . The same symbol is used to denote the set of jobs that are executed in this group. In the example above $G = \{j_1, \dots, j_m\}$.

4 Solution algorithm

In this section we present the dynamic programming algorithm that finds an optimal solution for $1 \mid r_j, p_j = p, pmnt \mid \sum w_i C_j$ problem. Idea of the algorithm is similar to the idea in non-preemptive case [2].

First, we suppose that $w_1 \geq w_2 \geq \dots \geq w_n$. Then, we consider sub-instances $I_k(s, \tau)$ of the initial instance. The set of jobs of $I_k(s, \tau)$ consists of such jobs $j \in N$ that $j < k$ (i.e., $w_j \geq w_k$) and $r_j \geq \tau$. The parameters of job j in the sub-instance are the following:

- processing time and weight are the same as in the initial instance;
- release date is equal to $\max\{s, r_j\}$.

Sub-instances $I_k(s, \tau)$ are constructed for $k = 1, 2, \dots, n$ and for each $s, \tau \in T$, where from the previous section $T = \{r_j + lp, j = \overline{1, n}, l = \overline{0, n}\}$. The total count of these sub-instances equals $O(n^5)$. By $\pi_k(s, \tau)$ we denote an optimal permutation schedule for sub-instance $I_k(s, \tau)$.

Let optimal permutations be constructed for sub-instances $I_{k-1}(s, \tau)$ for some k and all $s, \tau \in T$. Consider an optimal schedule $\pi_{k-1}(s, \tau)$ for fixed s and τ . Suppose that $r_k \geq \tau$ (otherwise, $\pi_k(s, \tau) = \pi_{k-1}(s, \tau)$).

Let $\pi_{k-1}(s, \tau)$ consist of m groups $G_1, G_2, \dots, G_m$. Construct schedules $\pi^{(0)}, \pi^{(1)}, \dots, \pi^{(m)}$ with the following procedure. Here, π_∞ denotes a schedule with infinite large cost value.

Procedure 2 (construct schedules $\pi^{(i)}$ for $i = 0, 1, \dots, m$)

- 1: **if** $i = 0$ **then**
- 2: $\pi^{(i)} := (k, \pi_{k-1}(s + p, \tau));$ // job k ends before all jobs
- 3: **else if** $i = m$ **then**
- 4: $\pi^{(m)} := (\pi_{k-1}(s, \tau), k);$ // job k ends after all jobs
- 5: **else**
- 6: /* job k ends after jobs from group $G_1, \dots, G_i$ and before jobs from groups $G_{i+1}, \dots, G_m$ */
- 7: let π' be a schedule of jobs from groups $G_1, \dots, G_i$ taken in the same order as in $\pi_{k-1}(s, \tau)$;
- 8: let S_k and C_k be a start and completion times of job k in sub-schedule (π', k) ;

```

9:   calculate  $S_k$  and  $C_k$  with Procedure 2;
10:  if  $\max\{r_j : j \in G_1 \cup \dots \cup G_i\} < \min\{r_j : j \in G_{i+1} \cup \dots \cup G_m\}$  then
11:     $\pi^{(i)} := (\pi', k, \pi_{k-1}(C_k, \tau'))$ , where  $\tau' = \min\{r_j : j \in G_{i+1} \cup \dots \cup G_m\}$ ;
12:  else
13:    if  $C_k = S_k + p$  (i.e., job  $k$  is fully executed after all jobs from  $G_i$ ) then
14:       $\pi^{(i)} := \pi_\infty$ ;
15:    else
16:      let  $\pi''$  be schedule of job from schedule  $\pi_{k-1}(s + p, \tau')$  that are completed
      after  $C_k$  taken in the same order, where  $\tau' = \min\{r_j : S_j > S_k\}$  ( $S_j$  is the
      start time of job in schedule  $\pi_{k-1}(s, \tau)$ );
17:       $\pi^{(i)} := (\pi', k, \pi'')$ ;
18:    end if
19:  end if
20: end if

```

Theorem 6 For each $k = 2, \dots, n$ we have $\pi_k(s, \tau) = \pi^{(i^*)}$, where $i^* = \arg \min_i F(\pi^{(i)})$.

Proof First of all, job k cannot be processed inside each group from schedule $\pi_{k-1}(s, \tau)$. This is true because the starter job in the group has smaller release date than S_k and larger weight. Since Corollary 1 from Theorem 3 this situation is prohibited for each optimal schedule. Therefore, job k can be processed on the following positions only:

- before all jobs from $\pi_{k-1}(s, \tau)$;
- after all jobs from $\pi_{k-1}(s, \tau)$;
- between each pair of joint groups from $\pi_{k-1}(s, \tau)$;

When we put the job between groups G_i and G_{i+1} , the left part of schedule $\pi^{(i)}$ can be obtained from schedule $\pi_{k-1}(s, \tau)$ by taking jobs from groups $G_1, \dots, G_i$ in the same order.

To construct the right part of schedule $\pi^{(i)}$ we need to consider the following cases:

1. If release dates of jobs from $G_1, \dots, G_i$ are less than release dates of jobs from $G_{i+1}, \dots, G_m$ then there exists sub-instance $I_{k-1}(C_k, \tau')$ on the upper level ($k-1$) with jobs from $G_{i+1}, \dots, G_m$ only, where $\tau' = \min\{r_j : j \in G_{i+1} \cup \dots \cup G_m\}$. In this case, schedule $\pi_{k-1}(C_k, \tau')$ is the desired right part.
2. If condition of Case 1 is not true and job k is fully processed after all jobs from G_i than there is no optimal schedule for the current sub-instance $I_k(s, \tau)$ where job k is processed between groups G_i and G_{i+1} . This holds because there exists a job j from $G_{i+1}, \dots, G_m$ with $r_j \leq S_k$ and $w_j \geq w_k$. Since Corollary 1 from Theorem 3 we can always improve such schedule.
3. If conditions of Cases 1 and 2 are not true then we consider schedule $\pi_{k-1}(s + p, \tau')$ where $\tau' = \min\{r_j : S_j > S_k\}$ (S_j is the start time of job in schedule $\pi_{k-1}(s, \tau)$). This schedule consists of jobs that are from set $G_\alpha, \dots, G_i, G_{i+1}, \dots, G_m$ for some $\alpha \geq 1$. Due to steps of Procedure 2, all of jobs from these groups are processed in schedule $\pi_{k-1}(s + p, \tau')$. Jobs from $G_\alpha, \dots, G_i$ are completed before jobs from $G_{i+1}, \dots, G_m$ in this schedule. The last job from $G_\alpha, \dots, G_i$ is completed at C_k . Therefore we can take the desired right part of $\pi_k(s, \tau)$ from this schedule.

◁

Finally, the algorithm for the problem is formulated as follows.

Algorithm 1

```

1: for each  $s, \tau \in T$ :  $\pi_1(s, \tau) = (1)$  if  $r_1 \geq \tau$ , otherwise  $\pi_1(s, \tau)$  is empty schedule;
2: for  $k = 2, 3, \dots, n$  do
3:   for each  $s, \tau \in T$  do
4:     construct auxiliary schedules  $\pi^{(i)}$  by Procedure 2;
5:     select the best of these auxiliary schedules and put it in  $\pi_k(s, \tau)$ ;
6:   end for
7: end for

```

On each step of the algorithm, $O(n^4)$ sub-instances are considered. For each sub-instance we try to put the current job k on $O(n)$ positions. When we put job k on some position we sequentially run Procedures 1 and 2 those complexity is $O(n^2)$ operations. The algorithm consists of n major steps. Therefore, the total complexity of the algorithm is $O(n^8)$ operations. Note that we do not need to enumerate all τ from set T . Necessary values of τ belong to set $r_1, \dots, r_n$. Therefore, we can reduce complexity of the algorithm to $O(n^7)$ operations.

5 Conclusion

In this paper, we propose a polynomial time algorithm that solves the minimizing total weighted completion time in preemptive equal-length job with release dates scheduling problem on a single machine $1 \mid r_j, p_j = p, pmnt \mid \sum w_i C_j$. Complexity of the algorithm is $O(n^7)$ operations.

References

1. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math., 287-326 (1979)
2. Ph. Baptiste, Scheduling Equal-Length Jobs on Identical Parallel Machines, Discrete Applied Mathematics, 103, 21-32 (2000)
3. W.E. Smith, Various optimizers for single-stage production, Nav. Res. Log. Q., 59-66 (1956)

Modeling Preemptive EDF and FP by Integer Variables

Enrico Bini

Abstract The design of any system can be modeled by an optimization problem, where a decision must be taken to maximize an overall utility function within some constraints (that can be physical, contractual, etc.). In hard real-time systems the constraints are specified by the deadlines that are set for the completion of tasks. However classic schedulability tests are formulated by algorithms that prevent a visualization of the feasible region of the designer choices.

In this paper we formulate the EDF and FP exact schedulability conditions on a single processor through a combination of linear constraints. We believe that this alternate representation is better suited for optimization and can trigger the development of more effective design methodologies for real-time systems.

1 Introduction

The design of any object or system can be formalized by an optimization problem: some decisions must be taken such that an overall utility function is maximized within some constraints. In real-time systems the feasibility constraint is expressed by a deadline requirement: a task is required to complete no later than a specified time (*deadline*) after its activation. Hence, it becomes crucial to find a representation of this constraint that is well suited for optimization techniques.

Classic feasibility tests for Fixed Priority (FP) and Earliest Deadline First (EDF) schedulers [14, 4] do not reveal the geometry of the feasible parameters. Hence some efforts must be dedicated to formulate the feasibility (deadline) constraint by alternate expressions that enable an efficient implementation of the system design by optimization techniques.

The schedule of jobs with known execution time is often represented by the start time of each job as variable. In fact once all the start times are given a schedule is unambiguously determined. Unfortunately this way of modeling becomes inappropriate when the number of variable associated to the start times becomes infinite or unknown, as in the following circumstances.

Enrico Bini
Scuola Superiore Sant'Anna, Pisa, Italy
E-mail: e.bini@sssup.it

1. If the lifetime of the system is potentially infinite and jobs are activated periodically (and not only a finite number of times) then we should have one start time variable per job. This leads to an infinite number of start times.
2. If the scheduling algorithm is preemptive then each job can be broken, by preempting jobs, into many contiguous chunks. This leads to an unknown number of start times (one for each non-preempted chunk).

Finally, the explicit assignment of the job start times may be inconsistent with the adopted scheduling algorithm, such as the EDF scheduling algorithm. EDF is indeed widely used in real-time systems to schedule jobs with deadline because of its optimality [17]: if it exists a job schedule that do not miss any deadline, then EDF will not miss any deadline.

Hence in this paper we show a representation of the EDF and FP schedulability condition in a way that is well suited for applying optimization techniques on the task parameters. We introduce our notation and terminology. We consider a set $\mathcal{T}$ of m periodic tasks executing on a single processor. Each task, denoted by τ_i , requires the execution of a job every *period* T_i . The j^{th} job of τ_i is often denoted by $\tau_{i,j}$. All the jobs of τ_i have the same *execution time* C_i . $\tau_{i,j}$ must complete not later than D_i time units after the activation. D_i is called *deadline* of the task τ_i . The *response time of a job* $R_{i,j}$ is the time that elapses from its activation to its completion. The response time of a task, simply called the *response time*, is $R_i = \sup_j R_{i,j}$. The *critical scenario* for τ_i is the scenario of task activations in which R_i is maximal. If tasks are scheduled by preemptive EDF or FP, the critical scenario for all tasks occurs when all tasks are activated simultaneously [17]. It is then convenient to set the instant when all the tasks start activating their first job equal to 0. Given these hypothesis, it follows that the *activation* of $\tau_{i,j}$ occurs at $a_{i,j} = (j-1)T_i$ and its *absolute deadline* is $d_{i,j} = (j-1)T_i + D_i$.

Task τ_i is also characterized by its *utilization* $U_i = \frac{C_i}{T_i}$. It represents the fraction of time that is required by the task. The sum of all the task utilizations $U = \sum_{i=1}^m U_i$ is the *total utilization* of the task set. It is straightforward to see that if $U > 1$, then some deadline is going to be missed because the processor is overloaded.

Sometime we denote the task τ_i also by the triplet (C_i, T_i, D_i) and the task set $\mathcal{T}$ by $(\mathbf{C}, \mathbf{T}, \mathbf{D})$, which are the vectors of computation times, periods, and deadlines, respectively.

All these parameters C_i , T_i , and D_i are real-valued. We extend the notion of least common multiple also to positive real numbers meaning that, given $a, b \in \mathbb{R}_+$ then

$$\text{lcm}(a, b) = \inf\{x \in \mathbb{R}_+ : \exists p, q \in \mathbb{N}_+ \ x = pa = qb\}$$

that is recursively defined also for a finite set of positive real number $\mathcal{A} = \{a_i\}_{i=1}^m$ as follows

$$\text{lcm}(\mathcal{A}) = \text{lcm}(a_i, \text{lcm}(\mathcal{A} \setminus \{a_i\}))$$

We consider both EDF and FP scheduling algorithms [17]. EDF assigns the highest priority to the job which has the earliest absolute deadline $d_{i,j}$. If two jobs have the same absolute deadline, tie is broken arbitrarily. In FP all jobs belonging to the same task have the same *task priority*. When using FP, we assume that tasks are indexed by decreasing priority: $i < j$ implies that τ_i has higher priority than τ_j . Tasks may not have the same priority.

1.1 Related Works

The space of feasible EDF deadlines was derived by Bini and Buttazzo [7] although the main result is justified based on geometric motivations and not formally proven. In the same paper also a convex restriction of the space of feasible deadlines is derived. Baruah and Bini [3] proposed a partitioning algorithm on multiprocessors, based on convenient formalization of the uniprocessor schedulability conditions. Hermant and George [12] proposed an effective simplification of the schedulability test using the simplex method.

Several authors [10, 2, 13] independently proposed different algorithms for computing the minimum deadline of a newly arrived task, assuming the existing task set is feasibly schedulable by EDF. The problem of these methods is that they can hardly be extended to reduce a set of arbitrary deadlines, but can only be applied to a single task at a time, following a given order, as suggested by [13].

When an FP scheduler is adopted, Bini and Buttazzo described the space of computation times, given periods and deadlines [6]. Seto et al [18] described the space of feasible periods. Bini and Di Natale [9] proposed a branch and bound algorithm for assigning the optimal feasible period assignment.

2 EDF schedulability analysis

The schedulability analysis is developed to check whether some deadlines can be missed or not. The classic way of performing schedulability analysis is based on the “demand-supply” approach. In simple words, by this approach it is required that the work demanded by the task set does not exceed the time supplied by the execution platform.

The necessary and sufficient schedulability condition for EDF was derived by Baruah et al. [4, 5].

Theorem 1 (Lemma 3 in [5]) *The task set $\mathcal{T} = \{(C_i, T_i, D_i) : i = 1, \dots, m\}$ is schedulable by EDF if and only if:*

$$\forall t \geq 0 \quad \sum_{i=1}^m \max \left\{ 0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor \right\} C_i \leq t \quad (1)$$

The necessary and sufficient condition of Eq. (1) cannot be used in practice, since it requires to check an infinite number of inequalities. Several works addressed the problem of reducing the number of instants where the inequality of Eq. (1) can be checked while preserving the necessity of the condition. First it can be observed that for any pair of adjacent absolute deadlines d_a and d_b , if Eq. (1) is true at d_a then it is also true $\forall t \in [d_a, d_b]$, because the left hand side of Eq. (1) remains constant whereas the right hand side increases. This allows to restrict the test to the set of all the absolute deadlines. The set of deadlines can be further reduced to a finite set by using linear upper bounds of the demand [5].

2.1 Integer Problem Formulation

Unfortunately, the condition expressed by Theorem 1 is not well suited to be used in optimizations, because the presence of the floor $\lfloor \cdot \rfloor$ operator breaks any property of

the constraints that is desirable for optimization (such as linearity). For this reason, some efforts have been devoted to the derivation of alternative way to formulate the necessary and sufficient condition for EDF schedulability.

The following Theorem provides a convenient way to formulate an equivalent condition of Theorem 1 that is expressed by a combination of linear constraints.

Theorem 2 *The task set $\mathcal{T} = \{(C_i, T_i, D_i) : i = 1, \dots, m\}$ is schedulable by EDF if and only if:*

$$\forall \mathbf{k} \in \mathbb{N}^m \setminus \{\mathbf{0}\} \quad \exists i \in I_{\mathbf{k}} \quad (T_i - C_i)k_i - \sum_{j \neq i} C_j k_j \geq T_i - D_i \quad (2)$$

where

$$I_{\mathbf{k}} = \{j : k_j \neq 0\}$$

is the set of non-zero indexes in $\mathbf{k}$.

Proof We will prove that Equations (1) and (2) are equivalent.

Eq. (1) $\Rightarrow$ Eq. (2). We are given a vector $\mathbf{k} \in \mathbb{N}^m$ different that $\mathbf{0} \in \mathbb{N}^m$ and we must find an index i in $I_{\mathbf{k}} = \{j : k_j \neq 0\}$ using the hypothesis of Eq. (1).

Since $\mathbf{k} \neq \mathbf{0}$ then $I_{\mathbf{k}} \neq \emptyset$. Let us define

$$\forall j \in I_{\mathbf{k}} \quad d_j = D_j + (k_j - 1)T_j$$

which is the absolute deadline of the k_j job of τ_j . Notice that from $k_j \geq 1$ it follows that $d_j \geq 0$.

We claim that the index i satisfying Eq. (2) is such that

$$d_i = \max_{j \in I_{\mathbf{k}}} \{d_j\} \quad (3)$$

This value is well defined because $I_{\mathbf{k}} \neq \emptyset$.

Now we exploit the Equation (1) for $t = d_i$. We have

$$\begin{aligned} & \sum_{j=1}^m \max \left\{ 0, \left\lfloor \frac{d_i + T_j - D_j}{T_j} \right\rfloor \right\} C_j \leq d_i \\ & \max \left\{ 0, \left\lfloor \frac{d_i + T_i - D_i}{T_i} \right\rfloor \right\} C_i + \sum_{j \neq i} \max \left\{ 0, \left\lfloor \frac{d_i + T_j - D_j}{T_j} \right\rfloor \right\} C_j \leq d_i \\ & \max \{0, k_i\} C_i + \sum_{j \neq i} \max \left\{ 0, \left\lfloor \frac{d_i + T_j - D_j}{T_j} \right\rfloor \right\} C_j \leq D_i + (k_i - 1)T_i \\ & (T_i - C_i)k_i - \sum_{j \neq i} \max \left\{ 0, \left\lfloor \frac{d_i + T_j - D_j}{T_j} \right\rfloor \right\} C_j \geq T_i - D_i \end{aligned} \quad (4)$$

Equation (4) is quite similar to Equation (2) that we want to prove. To conclude the demonstration we need to find a lower bound of $\max \left\{ 0, \left\lfloor \frac{d_i + T_j - D_j}{T_j} \right\rfloor \right\}$. We have

$$\begin{aligned} \forall j \in I_{\mathbf{k}} \quad \max \left\{ 0, \left\lfloor \frac{d_i + T_j - D_j}{T_j} \right\rfloor \right\} & \geq \left\lfloor \frac{d_i + T_j - D_j}{T_j} \right\rfloor \geq \\ & \geq \left\lfloor \frac{d_j + T_j - D_j}{T_j} \right\rfloor = k_j \end{aligned} \quad (5)$$

Similarly

$$\forall j \notin I_{\mathbf{k}} \quad \max \left\{ 0, \left\lfloor \frac{d_i + T_j - D_j}{T_j} \right\rfloor \right\} \geq 0 = k_j \quad (6)$$

Finally, from Equation (4) by using the lower bounds of Equations (5) and (6), it follows that for the index i selected such that Eq. (3) holds, we have

$$(T_i - C_i)k_i - \sum_{j \neq i} C_j k_j \geq (T_i - C_i)k_i - \sum_{j \neq i} \max \left\{ 0, \left\lfloor \frac{d_i + T_j - D_j}{T_j} \right\rfloor \right\} C_j \geq T_i - D_i$$

as required.

Eq. (2) $\Rightarrow$ Eq. (1). Given $t \geq 0$, we build the mapping $t \mapsto \mathbf{k} = (k_1, \dots, k_m) \in \mathbb{N}^m$ defined by

$$k_j = \max \left\{ 0, \left\lfloor \frac{t + T_j - D_j}{T_j} \right\rfloor \right\}$$

Given this mapping, the Equation (1) that needs to be proved can be rewritten as

$$\forall t \geq 0 \quad \sum_{j=1}^m k_j C_j \leq t \quad (7)$$

For all the $t \geq 0$ such that $\mathbf{k} = \mathbf{0}$ we have

$$\sum_{j=1}^m k_j C_j = 0 \leq t$$

On the other hand, for all the $t \geq 0$ such that $\mathbf{k} \neq \mathbf{0}$ we can invoke Equation (2) from which it follows that we have an index $i \in I_{\mathbf{k}}$ such that

$$\begin{aligned} (T_i - C_i)k_i - \sum_{j \neq i} C_j k_j &\geq T_i - D_i \\ \sum_{j=1}^m C_j k_j &\leq (k_i - 1)T_i + D_i = \left(\left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor - 1 \right) T_i + D_i \leq \frac{t - D_i}{T_i} T_i + D_i \\ \sum_{j=1}^m C_j k_j &\leq t \end{aligned}$$

Hence Equation (7) is demonstrated and the Theorem is proved.

Theorem 2 formulates the EDF schedulability by introducing the integers $\mathbf{k}$. However the “physical” interpretation of Eq. (2) is lost, whereas Eq. (1) has the clear interpretation of requiring that the amount of demand should not exceed the time available to schedule the tasks. To extract a posteriori the physical interpretation of Theorem 2 then we rewrite Eq. (2) as follows

$$\forall \mathbf{k} \in \mathbb{N}^m \setminus \{\mathbf{0}\} \quad \exists i : k_i \geq 1 \quad \mathbf{C} \cdot \mathbf{k} \leq (k_i - 1)T_i + D_i = d_{i,k_i}$$

that is equivalent to

$$\forall \mathbf{k} \in \mathbb{N}^m \setminus \{\mathbf{0}\} \quad \mathbf{C} \cdot \mathbf{k} \leq \max_{k_i \geq 1} \{d_{i,k_i}\} \quad (8)$$

Basically Eq. (8) asserts that the task set $\mathcal{T}$ is schedulable by EDF if and only if for all possible number of jobs of each task (represented by the vector $\mathbf{k}$), the amount of all required computations $\mathbf{C} \cdot \mathbf{k}$ does not exceed the largest absolute deadline.

In the constrained deadline case (i.e. $\forall i \quad D_i \leq T_i$) the schedulability test can be simplified as reported in the following result.

Lemma 1 *The task set $\mathcal{T} = \{(C_i, T_i, D_i) : i = 1, \dots, m\}$ with constrained deadlines is schedulable by EDF if and only if:*

$$\forall \mathbf{k} \in \mathbb{N}^m \setminus \{\mathbf{0}\} \quad \exists i = 1, \dots, m \quad (T_i - C_i)k_i - \sum_{j \neq i} C_j k_j \geq T_i - D_i \quad (9)$$

Proof We proceed by showing that if $\forall i \quad D_i \leq T_i$ then Equations (2) and (9) are equivalent.

Eq. (2) $\Rightarrow$ Eq. (9). This is the simpler implication. In fact since $I_{\mathbf{k}} \subseteq \{1, \dots, m\}$, then the same index i that makes Eq. (2) true, makes true also Eq. (9).

Eq. (9) $\Rightarrow$ Eq. (2). Given $\mathbf{k}$, let i be the index in $\{1, \dots, m\}$ that satisfies Eq. (9). Let us assume, by absurd, that $i \notin I_{\mathbf{k}}$ that means $k_i = 0$. From Eq. (9) we have

$$\sum_{j=1}^m C_j k_j \leq D_i - T_i \leq 0 \quad \Rightarrow \quad \mathbf{k} = \mathbf{0}$$

that is impossible from the hypothesis. Hence the lemma is proven.

From now on we assume to have constrained deadlines to simplify the presentation since the arbitrary deadline model does not present significant additional difficulties.

The schedulability condition expressed in this way is not practical, since it requires to test on an infinite set (i.e. $\mathbb{N}^m \setminus \{\mathbf{0}\}$). However there are methods to reduce the test to a finite set without losing necessity [7]. We do not discuss here these methods any further.

The schedulability condition of Eq. (9), can also be seen as a covering problem. We define

$$\text{dom}K_i = \{\mathbf{k} \in \mathbb{Z}^m : \sum_{j \neq i} C_j k_j - (T_i - C_i)k_i \leq D_i - T_i\} \quad (10)$$

that basically is the set of integers that falls in an half-space. Then the schedulability condition of Lemma 1 is equivalent to the following

$$\mathbb{N} \setminus \{\mathbf{0}\} \subseteq \bigcup_{i=1}^m \text{dom}K_i \quad (11)$$

3 FP schedulability analysis

Here we also report briefly a similar representation of the schedulability constraint that applies to FP schedulers.

In preemptive FP schedulers any task is not affected by the lower priority tasks because lower priority jobs can never execute when some higher priority job has not yet finished. Hence the FP schedulability analysis is developed per task and the schedulability of the entire task set $\mathcal{T}$ is ensured by the simultaneous verification of the schedulability condition for each task. For this reason, without loss of generality, in this section we consider only the schedulability of the lowest priority task τ_m . Moreover we examine only the constrained deadline case, since the extension to the arbitrary deadline case follows standard techniques [15].

Two main classes of tests exist for FP schedulability analysis: the response time analysis [11,14,1] and the time demand analysis [16].

In the response time analysis [11,14,1] the following algorithm

$$\begin{cases} R_m^{(0)} & \leftarrow C_m \\ R_m^{(k+1)} & \leftarrow C_m + \sum_{i=1}^{m-1} \left\lceil \frac{R_m^{(k)}}{T_i} \right\rceil C_i \end{cases} \quad (12)$$

is iterated until the sequence $R_m^{(k)}$ converges to the response time $R_m \leq D_m$ of τ_m (in this case τ_m is schedulable) or at some iteration it exceeds the deadline D_k (in this case τ_m is not schedulable).

The time demand analysis [16] does not go through the computation of the response time. Basically it looks for an instant within 0 and D_m that is large enough to accommodate both the computation time C_m of τ_m and the maximum number of interferences from higher priority tasks. This idea is represented by the following Theorem

Theorem 3 (Theorem 1 in [16]) *The task τ_m with constrained deadline $D_m \leq T_m$ is schedulable by FP if and only if*

$$\exists t \in [0, D_m] \quad C_m + \sum_{i=1}^{m-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t \quad (13)$$

Both the approaches has the same defect of Theorem 1: the presence of a discontinuous operator (the ceiling $\lceil \cdot \rceil$ in this case) makes difficult to use the schedulability test as a description of the feasible region.

Fortunately, Seto et al. [18] found an alternate expression of the feasibility region. Below we report their result, derived in the implicit deadline hypothesis, extended to constrained deadline case.

Theorem 4 (Proposition 2.1 in [18]) *The task τ_m is schedulable by FP if and only if*

$$\exists \mathbf{k} \in \mathbb{N}^{m-1} \quad \begin{cases} C_m + \sum_{j=1}^{m-1} C_j k_j \leq D_m \\ C_m + \sum_{j=1}^{m-1} C_j k_j \leq k_i T_i \quad i = 1, \dots, m-1 \end{cases} \quad (14)$$

4 Future Works

In this paper we have illustrated methods for expressing the schedulability constraints (both EDF and FP) by means of a combination of linear constraints. The next step will be the formulation of the design goal by means of a utility function to be maximized, and the optimal solution of the design problem, following similar approaches as proposed by Seto et al. [19] or Bini and Cervin [8].

References

1. Neil C. Audsley, Alan Burns, Mike Richardson, Ken W. Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.
2. Patricia Balbastre, Ismael Ripoll, and Alfons Crespo. Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 65–74, Dresden, Germany, July 2006.
3. Sanjoy Baruah and Enrico Bini. Partitioned scheduling of sporadic task systems: an ILP-based approach. In *Conference on Design and Architectures for Signal and Image Processing*, Bruxelles, Belgium, November 2008.
4. Sanjoy K. Baruah, Rodney Howell, and Louis Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.
5. Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 182–190, Lake Buena Vista (FL), U.S.A., December 1990.
6. Enrico Bini and Giorgio C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, November 2004.
7. Enrico Bini and Giorgio C. Buttazzo. The space of EDF deadlines; the exact region and a convex approximation. *Real-Time Systems*, 2008.
8. Enrico Bini and Anton Cervin. Delay-aware period assignment in control systems. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, Barcelona, Spain, December 2008.
9. Enrico Bini and Marco Di Natale. Optimal task rate selection in fixed priority systems. In *Proceedings of the 26th IEEE Real-Time Systems Symposium*, pages 399–409, Miami (FL), U.S.A., December 2005.
10. Giorgio Buttazzo and Fabrizio Sensini. Optimal deadline assignment for scheduling soft aperiodic task in hard real-time environments. *IEEE Transactions on Computers*, 48(10):1035–1052, October 1999.
11. Paul K. Harter. Response times in level-structured systems. Technical Report CU-CS-269-84, Department of Computer Science, University of Colorado, USA, 1984.
12. Jean-François Hermant and Laurent George. A norm approach for the partitioned edf scheduling of sporadic task systems. In *Proceedings of the 21st Euromicro Conference on Real-Time Systems*, Dublin, Ireland, July 2009.
13. Hoai Hoang, Giorgio Buttazzo, Magnus Jonsson, and Stefan Karlsson. Computing the minimum EDF feasible deadline in periodic systems. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 125–134, Sydney, Australia, August 2006.
14. Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, October 1986.
15. John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadline. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 201–209, Lake Buena Vista (FL), U.S.A., December 1990.
16. John P. Lehoczky, Lui Sha, and Ye Ding. The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pages 166–171, Santa Monica (CA), U.S.A., December 1989.
17. Chung Laung Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.
18. Danbing Seto, John P. Lehoczky, and Lui Sha. Task period selection and schedulability in real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 188–198, Madrid, Spain, December 1998.
19. Danbing Seto, John P. Lehoczky, Lui Sha, and Kang G. Shin. On task schedulability in real-time control systems. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 13–21, Washington (DC), U.S.A., December 1996.

Home health care problem

An extended multiple Traveling Salesman Problem

Yannick Kergosien · Christophe Lenté ·
Jean-Charles Billaut

Abstract This paper deals with the routing problem of health care staff in a home health care problem. Given a list of patients needing several cares, the problem is to assign cares to care workers. Some cares have to be performed by several persons and some cares cannot be performed with others. If a patient needs several cares, he may want to be treated by the same person. Moreover, some skills constraints and time windows have to be satisfied. We show that this problem is equivalent to a multiple traveling salesman problem with time windows (mTSPWT) with some specific constraints. For solving this problem, we propose an integer linear program with some technical improvements.

1 Introduction

The Home Health Care (HHC) demand is growing rapidly in France as well as in several other countries [19]. It includes services such as nursing, physical therapy, speech therapy, medical social services, medical visits, house cleaning, home life aides, old people assistance, etc. This development is mainly due to economic factors [1], congestion of hospitals, preferences of the patients [21], ageing of the population [2]. The HHC aims to reducing the number of patients in hospitals, to improving the quality of care, and minimizing the costs of health [17]. Many papers deal with the HHC and its advantages (see [10] and [23]).

A new problem of organization appears: how is it possible to coordinate care workers while ensuring good quality of care. In this paper, we are only interested in the routing problem of health care staff, subject to some coordination problems in the residences of patients. Some cares may need simultaneously several care workers, for example a nurse and a doctor, or two care assistants to help to get up an impotent person. Conversely, some care workers cannot work simultaneously, for example a nurse and a physical therapist. It is better if patients always see the same care workers (same nurse,

Yannick Kergosien, Christophe Lenté, Jean-Charles Billaut
Université François Rabelais Tours,
Laboratoire d'Informatique,
64 av. Jean Portalis, 37200 Tours.
E-mail: {yannick.kergosien|christophe.lente|jean.billaut}@univ-tours.fr

same doctor, etc.) as long as possible. We show that this problem is equivalent to a multiple traveling salesman problem with time windows (mTSPWT) and some special constraints. We propose an integer linear programming formulation (ILP) with some cuts for solving it.

2 Literature review

Some articles related to home health care workers routing problem already exist in the literature. In [5], the authors study the scheduling and routing of home health care nursing in Alabama and develop a spatial decision support system. They build a heuristic to construct the routes of each nurse, taking unavailabilities constraints into account. In a paper of V. De Angelis [9], the main problem is not a problem of routing but of allocating resources (doctors, nurses, social assistance and caring for patients) within a given budget, in order to maximize the number of patients delivered. To solve this problem, the author formulates a linear program model. Another linear integer model is developed in [7]. This model aims to solve the human resource short term planning in order to satisfy each patient on time. T. Fahle, in [13], studied the problem of vehicle routing and staff rostering aspects in home health care. He described a model data of generic home health care problem software, and presented a method to solve this problem. In our study, we are not interested in the rostering problem. In [8], the problem of routing home health care is also studied by considering two types of nursing, part time and full time, with a different cost per hour. They define a mixed integer program taking into account the lunch pauses and they present a basic heuristic with the objective of minimizing the total cost. A similar problem is tackled in [6] with nurses having different skills. Here, the objective is not only to minimize the total cost but a weighted sum of the total travel time plus a sum of several penalties like the violation of time windows or of patients preferences. The heuristic developed by the authors for solving this problem is divided into two parts: (1) build a set of patients to serve for each nurse and (2) find an optimal sequencing for each set of patients. A very similar problem is studied in [11] in a Swedish environment, but the objective is to minimize the travel time and the waiting time of patients. The authors solve this problem using a set partitioning model with two types of variables (some for assigning a staff member to a schedule, some for assigning a staff member to a visit with a vehicle). A matching approach is used iteratively for finding a solution. They described also the development of a decision support system called LAPS CARE to eliminate the manual planning of home care unit assignments. In another paper [12], the same authors presented and discussed some results and experiences from two local government organizations and from the Laps Care. They conclude on an improvement of operational efficiency and of the quality of home care for elderly citizens. In [3], the same problem is solved by a Particle Swarm Optimization algorithm (population-based evolutionary algorithm with a local search algorithm). The first algorithm explores the solution space globally whereas the local search explores the neighborhood of a solution, using improvement procedures like swap or insertion. Finally, one of the most recent articles dealing with the problem of routing home health care people is [4]. The authors take the most possible constraints into account but they consider only one type of care workers and do not take cares exclusions into account. They formulate an integer linear model and compare two commercial solvers (Cplex and Lingo). However, the computation time is quite important even on small instances.

3 Problem presentation

We are interested in the routing problem of the care staff. Several doctors, nurses, physical therapists, care assistants, housekeepers, etc., are available to treat some patients at home, but under restrictions on working hours or locations. The patients to be treated are dispersed geographically and require possibly several cares. A care is characterized by a duration, an earliest and a latest starting times and one needed category of care workers (a nurse or a doctor or etc.) but a care can also be pre-assigned to a specific person, for example because he is the family doctor or the nurse of the patient. A care can also require several care workers. In this case, we consider that there are as many cares as the number of care workers needed and that these cares must begin at the same time or with known delays. Conversely, other cares can be in disjunction: they cannot take place simultaneously. For example, a speech therapist and a doctor cannot work together, either the doctor comes first or it is the speech therapist. Some cares have to be performed by the same person (not necessary pre-assigned), for example if a nurse has to come at the morning and at the afternoon, the patient may prefer that the same nurse makes the two cares, whoever is the nurse.

We consider a set $M = \{1, \dots, m\}$ of care workers, c_k the skill of person k (doctor, nurse, etc.). We have to assign to each person an ordered list of cares to perform. Each person k must leave and return to its domicile (or office) D_k , with D the set of domiciles. A person works only during its shift noted by a time window $[e_{D_k}, l_{D_k}]$ which respectively represents the earliest date and latest date of work of person k on its domicile D_k . In addition, a person k may require one or more breaks during his service. We will see that this break may be seen like a fictitious pre-assigned care.

There is a set of patients to treat at home possibly several times by perhaps several people. We define a service by only one care to one patient performed by one person. A patient requiring a treatment involving three care workers will be represented by three synchronized services. We define a set $S = \{1, \dots, n\}$ of services. A service i is characterized by a required skill q_i , a time window $[e_i, l_i]$ during which the service to perform must begin and a duration of treatment p_i . The travel time between two locations of services i and j is denoted by $d_{i,j}$. We note $\Omega = \{\omega_1, \dots, \omega_r\}$ the set of treatments requiring more than one care worker. Each element ω_t is in fact a set of synchronized services, the synchronization between two services i and j is given by a delay $\delta_{i,j}$: service i must begin $\delta_{i,j}$ time units after service j . We note $\Phi = \{\phi_1 \dots \phi_o\}$ the set of set of services which must not be performed simultaneously, and $\Gamma = \{\gamma_1 \dots \gamma_s\}$ the set of set of services which must be performed by the same person. If a service i is pre-assigned we note π_i the number of the pre-assigned people, if i is not pre-affected, π_i is set to 0. We can therefore represent a pause by a pre-assigned service to the person concerned.

This problem is clearly similar to a traveling salesman problem with time windows and some special constraints. The TSP [14] has been often studied in the literature. This type of problem with a fixed number of traveling salesman is called m -TSP. It is a generalization of the classical TSP. In [22], the author enumerates the possible extensions of this problem (time windows, fixed number of traveling salesmen, maximum or minimum number of vertices to visit, etc.), its practical applications, some formulations in linear programming and some heuristic or exact methods to solve these problems. Considering the time windows version, check if an instance of m -TSP with time windows is feasible is NP-complete [18]. One of the first ILP formulation for TSP was made by [16]. This formulation has been extended to several traveling salesmen by

[20], the authors introduce also a branch-and-bound algorithm to solve the m -salesman traveling problem by using linear relaxations. In [15], the authors propose extending these linear models by integrating a minimum and maximum number of vertices to visit for each salesman in the case of multiple depots.

The HHC problem is thus equivalent to a m -TSP with some special constraints on a complete directed graph noted $G = \{V; A\}$ with: $V = S \cup D$ the set of vertices (one vertex for each service and for each depot), and A the set of arcs, the length of an arc between vertices i and j being the sum of the travel time $d_{i,j}$ between i and j the duration p_i of service i , if $i \in S$. Finally, we note $cost_{i,j}^k$ the cost of passage of the person on the arc (i, j) . The goal is to find the routes of each person such as each vertex is visited by exactly one person and the constraints are satisfied. The objective is to minimize the total cost of the routes followed by each care worker.

4 ILP

4.1 Linear programming model

We propose now an exact resolution of this problem by using an integer linear programming formulation (ILP).

This formulation uses the three following types of variables:

$$\forall i, j \in V \ (i \neq j), \forall k \in M : \quad x_{i,j}^k = \begin{cases} 1 & \text{if person } k \text{ performs service } i \text{ and then } j \\ 0 & \text{otherwise} \end{cases}$$

$$\forall \phi_h \in \Phi, \forall i, j \in \phi_h \ (i \neq j) : \quad y_{i,j} = \begin{cases} 1 & \text{if service } i \text{ is performed before } j \\ 0 & \text{otherwise} \end{cases}$$

$$\forall i \in V : \quad t_i \in [e_i, l_i] : \text{starting date of service } i$$

The first type of variables is used to define the routes of care workers. Thanks to the time windows we can set some variables to the value 0 (if $e_i + p_i + d_{i,j} > l_j$ then $\forall k \in M : x_{i,j}^k = 0$). The second type is used to ensure the disjunctions between services which cannot be performed at the same time. The last type of variables is used for the satisfaction of all time windows.

The problem can be defined by 11 sets of constraints:

$$\forall j \in S : \sum_{\forall k \in M} \sum_{\forall i \in V \setminus j} x_{i,j}^k = 1 \quad (1)$$

$$\forall k \in M, \forall i \in V : \sum_{\forall j \in V \setminus i} x_{j,i}^k = \sum_{\forall j \in V \setminus i} x_{i,j}^k \quad (2)$$

$$\forall k \in M : \sum_{\forall i \in V \setminus D_k} x_{D_k,i}^k \leq 1 \quad (3)$$

$$\forall k \in M, \forall i \in S : x_{D_k,i}^k (e_{D_k} + d_{D_k,i}) \leq t_i \quad (4)$$

$$\forall i, j \in S, i \neq j : t_i + d_{i,j} + p_i \leq t_j + HV(1 - \sum_{\forall k \in M} x_{i,j}^k) \quad (5)$$

$$\forall k \in M, \forall i \in S : t_i + d_{i,D_k} + p_i \leq x_{i,D_k}^k (l_{D_k} - L) + L \quad (6)$$

$$\forall \omega_h \in \Omega, \forall i, j \in \omega_h (i \neq j) : t_i = t_j + \delta_{i,j} \quad (7)$$

$$\forall \phi_h \in \Phi, \forall i, j \in \phi_h (i \neq j) : y_{i,j} = 1 - y_{j,i} \quad (8)$$

$$\forall \phi_h \in \Phi, \forall i, j \in \phi_h (i \neq j) : t_i + p_i \leq t_j + HV(1 - y_{i,j}) \quad (9)$$

$$\forall \gamma_h \in \Gamma, \forall i, j \in \gamma_h (i \neq j), \forall k \in M : \sum_{\forall l \in V \setminus i} x_{l,i}^k = \sum_{\forall l \in V \setminus j} x_{l,j}^k \quad (10)$$

$$\forall i \in S / \pi_i \neq 0 : \sum_{\forall l \in V \setminus i} x_{l,i}^{\pi_i} = 1 \quad (11)$$

Constraints (1) impose that each service is performed by exactly one care worker. Constraints (2) ensure the continuity of routes: when a person enters in a vertex, he has to exit. Constraints (3) impose that each care worker will exit from its domicile only once (remarks: $\forall k, k' \in M, k \neq k', \forall i \in V : x_{D_k,i}^{k'} = x_{i,D_k}^{k'} = 0$). Constraints (4) define the earliest starting time of work of the care worker k if he performs service i . Constraints (5) formulate the travel times between two vertices (HV is a notation to indicate a value presumably infinite). The constraints (6) impose to each care worker k to come back to its domicile before its date of end of work (L is a constant value equal to $\max_{k \in M} (l_{D_k})$, the greatest date of end of work). Constraints (7) link the synchronized services, considering the time lags $\delta_{i,j}$. Constraints (8) and (9) deal with the services that are in disjunction. Constraints (8) impose that service i is performed before service j , or the contrary. If service i is performed before j then the constraints (9) ensure that the treatment of j does not begin before the end of treatment of i . However, it is unnecessary to add this constraint if $l_i + p_i < e_j$ or $l_j + p_j < e_i$. Constraints (10)) express that some services must be performed by the same care worker. Constraints (11) assign all care workers. The competence of the services he performed is ensured by the following pre-processing:

$$\text{if } q_i \neq c_k \text{ then } \forall j \in V, (i \neq j) : x_{i,j}^k = x_{j,i}^k = 0$$

The objective function of the ILP is to minimize the total travelling cost:

$$\min \sum_{\forall k \in M} \sum_{\forall i \in V} \sum_{\forall j \in V \setminus i} cost_{i,j}^k x_{i,j}^k \quad (12)$$

4.2 Technical improvements

In order to improve the resolution time, many techniques have been tested. After several experiments, the following techniques are used.

First, some cuts can be added according to the time windows. Two services with two overlapping time windows require two different care providers. This type of cut can be formulated as follows:

$$\begin{aligned} \forall i, j \in S \ i \neq j : \\ \text{if } e_i + p_i + d_{i,j} > l_j \text{ and } e_j + p_j + d_{j,i} > l_i \text{ then } \forall k \in M : \\ \sum_{\forall l \in V \setminus i} x_{l,i}^k + \sum_{\forall l \in V \setminus j} x_{l,j}^k \leq 1 \end{aligned}$$

In the same way, the cares which require several people are split into several services that cannot be assigned to the same person. We present here the cuts concerning the care requiring 2 and 3 people, but it can be easily extended to more people.

$$\begin{aligned} \forall \omega_h \in \Omega : \\ \text{if } \|\omega_h\| = 2 \text{ then } \forall i, j \in \omega_h \ (i \neq j), \forall k \in M : \\ \sum_{\forall l \in V \setminus i} x_{l,i}^k + \sum_{\forall l \in V \setminus j} x_{l,j}^k \leq 1 \\ \text{if } \|\omega_h\| = 3 \text{ then } \forall i, j, o \in \omega_h \ (i \neq j \neq o), \forall k \in M : \\ \sum_{\forall l \in V \setminus i} x_{l,i}^k + \sum_{\forall l \in V \setminus j} x_{l,j}^k + \sum_{\forall l \in V \setminus o} x_{l,o}^k \leq 1 \end{aligned}$$

Finally, to avoid the exploration of identical solutions, we added a term in the objective function. Indeed, each route of two identical care workers (same domicile, same skill, same work time and same cost) can be exchanged without changing the value of the objective function. As there are often identical care workers in real instances, many solutions with the same cost are explored. In order to improve the exploration significantly, we define a new objective function with two criteria to optimize in a lexicographical way. The first criterion is the previously defined objective function. The second criterion is a sort of corrective term that helps avoiding identical solutions, it tends to assign the longest routes to care workers with the smallest numbers. We thus define the new following objective function.

$$\min \alpha \times \sum_{\forall k \in M} \sum_{\forall i \in V} \sum_{\forall j \in V \setminus i} cost_{i,j}^k x_{i,j}^k + \sum_{\forall k \in M} k \sum_{\forall i \in V} \sum_{\forall j \in V \setminus i} d_{i,j} x_{i,j}^k$$

The term α is supposed to be greater than the maximal value of the corrective term, so that the lexicographic optimization is guaranteed.

5 Computational experiments

The ILP has been tested on random generated instances with the solver Cplex. We have considered three types of instances with 1, 2 and 3 skills and with 10, 20, 30 and 40

services. For each type of instance, we have generated 200 difficult instances. Duration of cares belongs to the interval [10 minutes, 1 hour]. The length of time windows is between half an hour and three hours. On average 20% of services are synchronized ones, 20% of services are in exclusion, and 20% of services must be performed by the same care worker. The number of care workers is calculated such that they are occupied on average 70% of the day to obtain difficult instances and sometimes infeasible. All care workers have the same associated cost ($cost_{i,j}^k = d_{i,j}$). The various locations are randomly generated on a 100×100 square and the travel time $d_{i,j}$ is set to the Euclidean distance between these two locations. The experiments have been conducted on a Pentium IV at 2.8 GHz with 512 Mo of RAM. The time of resolution was limited to 10 minutes only. Table 1 presents the average time to solve an instance, the standard deviation of time resolution, the percentage of instances solved for each type of instance, and the GAP of Cplex for unsolved instances. The Gap is defined as $(f(ILP) - f(LP))/f(LP)$ where $f(ILP)$ is the best integer solution found by Cplex and $f(LP)$ is the optimal linear solution. We compare in this table two ILP: with and without technical improvements. We achieved these results after trying to set the parameters of CPLEX. The resolution settings of CPLEX were not changed except the level of probing at the highest level.

Table 1 Computational experiments

Number of services	Number of skills	Without Technical improvements				With Technical improvements			
		CPU(s)	σ	$\tau(\%)$	GAP(%)	CPU (s)	σ	$\tau(\%)$	GAP(%)
10	3	0,03	0,02	100,0	-	0,03	0,02	100,0	-
10	2	0,06	0,07	100,0	-	0,05	0,06	100,0	-
10	1	0,45	0,81	100,0	-	0,42	0,58	100,0	-
20	3	1,14	3,9	100,0	-	1,65	7,14	100,0	-
20	2	10,1	55,2	100,0	-	8,8	40,5	100,0	-
20	1	220,6	253,6	58	18,2	181,7	229,2	58	16,1
30	3	41,0	106,1	90,0	9,1	46,8	132,3	89,5	9,7
30	2	144,1	200,1	63,5	9,8	136,2	184,6	66,5	10,0
40	3	167,9	229,1	52,0	7,0	107,4	142,8	57,0	5,8

The number of skills is strongly connected to the time of resolution because the number of possible solution increases when the number of skills decreases. Lastly, the time of resolution, as well as the standard deviation, is decreased thanks to the technical improvements but also the number of unsolved instances and the GAP.

6 Conclusion

We have presented a routing problem in a very general context of Home Health Care. It involves care workers with various skills and patients with various needs. The patients must be visited according to times windows. This problem can be assimilated to a multi-traveling salesman problem with time windows in which some cities must be visited by the same traveler or by a specific traveler. Some cities cannot be visited by the same traveler and some cities must be visited simultaneously or with a known delay. The proposed integer linear program is not able to deal with instances of real size but it appears in the tests that it gives rapidly a good solution which is promising to apply technique like column generation.

References

1. Afrite A., Com-Ruelle L., Or Z., Renaud T.: L'hospitalisation à domicile, une alternative économique pour les soins de suite et de réadaptation. *Question d'économie de la santé*, n119, (2007).
2. Ahlner-Elmqvist M., Jordhoy M.S., Bjordal K., Jannert M., Kaasa S.: Characteristics and Quality of Life of Patients Who Choose Home Care at the End of Life. *Journal of Pain and Symptom Management*, Volume 36, Issue 3, (2008).
3. Akjiratikarl C., Yenradee P., Drake P.R.: PSO-based algorithm for home care worker scheduling in the UK. *Computers & Industrial Engineering* 53, 559-583, (2007).
4. Ben Bachouch R., Fakhfakh M., Guinet A., Hajri-Gabouj S.: Planification de la tournée des infirmiers dans une structure de soins à domicile. *Gestion et Ingénierie des Systèmes Hospitaliers*, 4ème conférence GISEH EPFL-Lausanne, (2008).
5. Begur S.V., Miller D.M., Weaver J.R.: An integrated spatial Decision Support System for scheduling and routing home health care nurses. *Institute of operations research and the management science*, 35-48, (1997).
6. Bertels S., Fahle T.: A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem *Computers & Operations Research* 33, 2866-2890, (2006).
7. Borsani V., Matta A., Beschi G., Sommaruga F.: A home care scheduling model for human resources. *Service Systems and Service Management, International Conference*, 449-454, (2006).
8. Cheng E., Rich J. L.: A home care routing and scheduling problem. Technical Report TR98-04, Department of Computational And Applied Mathematics, Rice University (1998).
9. De Angelis V.: Planning Home Assistance for AIDS Patients in the City of Rome, Italy, *Institute of operations research and the management science* 28, 75-83 (1998).
10. Duke M.: Home Care. *International Encyclopedia of Public Health*. Academic Press, Oxford, 455-460, (2008).
11. Eveborn P., Flisberg P., Ronnqvist M.: LAPS CARE-an operational system for staff planning of home care. *European Journal of Operational Research* 171, 962-976, (2006).
12. Eveborn P., Ronnqvist M., Einarsdottir H., Eklund M., Liden K., Almroth M.: Operations Research Improves Quality and Efficiency in Home Care. *Interfaces*, Vol. 39, No. 1, 18-34, (2009).
13. Torsten Fahle: Production and Transportation Planning Modeling Report. Report, (2001).
14. Gutin G., Abraham P.: The Traveling Salesman Problem and Its Variation. Punne, Springer Verla, (2002).
15. Kara I., Tolga B.: Integer linear programming formulations of multiple salesman problems and its variations. *European Journal of Operational Research*, 174, 1449-1458, (2006).
16. Miller C.E., Tucker A.W., Zemlin R.A.: Integer programming formulation of travelling salesman problems. *J. Assoc. Comput. Mach.*, 7, 326-329, (1960).
17. Raffy N.: L'hospitalisation à domicile : une alternative économique = Home care : an economic alternative. *Concours médical* ISSN 0010-5309 Coden Comeao. Vol. 116, n13, (1994).
18. Savelsbergh M.W.P.: Local search in routing problems with time windows. *Ann. Oper. Res.*, 4, 285-305, (1985).
19. Shepperd S. and Iliffe S.: Hospital at home versus inpatient hospital care. *The Cochrane Database of Systematic Reviews* (2005).
20. Svetska J.A., Huckfeldt V.E.: Computational experience with an m-salesman travelling salesman algorithm. *Management Sci.*, 19, 790-799, (1973).
21. Sentilhes-Monkam A.: L'hospitalisation à domicile et la prise en charge de la fin de vie : le point de vue des patients et de leurs proches. *Santé publique - 3* (Vol. 18), ISSN 0995-3914, (2006).
22. Tolga B.: The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34, 209-219, (2006).
23. Van den Berg B., Hassink W.H.J.: Cash benefits in long-term home care. *Health PolicyIn Press*, (2008).

A GRASP approach for the delay-constrained multicast routing problem

Ying Xu • Rong Qu

Abstract The rapid development of real-time multimedia applications requires Quality of Service (QoS) based multicast routing in underlying computer networks. The constrained minimum Steiner tree problem in graphs as the underpinning mathematical model is a well-known NP-complete problem. In this paper we investigate a GRASP (Greedy Randomized Adaptive Search Procedure) approach with VNS (Variable Neighborhood Search) as the local search strategy for the Delay-Constrained Least-Cost (DCLC) multicast routing problems. A large number of simulations carried out on the benchmark problems in the OR-library and a group of randomly generated graphs demonstrate that the proposed GRASP algorithm with VNS is highly efficient in solving the DCLC multicast routing problem. It outperforms other existing algorithms and heuristics in the literature.

1 Introduction

Multicast routing is a mechanism which transfers information from a source to a group of destinations simultaneously. With the increasing development of numerous multicast network applications (e.g. E-learning, E-commerce, video-conferencing), the underlying computer network requires multicast routing with certain QoS (Quality of Service) constraints. One example is that many of these real-time applications can tolerate only a bounded end-to-end delay. Other QoS constraints in reality include cost, bandwidth, delay variation, lost ratio and hop count, etc. Multicast QoS routing has received significant research attention in the area of computer networks and algorithmic network theory [1-3]. This paper is concerned with two of the most important QoS demands, the total cost of the multicast tree from the source to all the destinations and the end-to-end delay bound for the total delay from the source to any destination in the multicast group.

Multicast routing problems can be reduced to Minimum Steiner Tree Problems in Graph (MStTG) [4]. Generally, given an undirected graph $G = (V, E)$, where V is a set of nodes, E is a set of edges, and a subset of nodes $D \subseteq V$, a Steiner tree is a tree which connects all the nodes in D using a subset of edges in E . Extra nodes in $V \setminus D$ may be added to the Steiner tree, called the Steiner nodes. The MStTG problem is to search for a minimal Steiner tree with respect to the total edge costs $c(e)$, $e \in E$, which has been proven to be NP-complete [5]. The Delay-Constrained Least-Cost (DCLC) multicast routing problem searches for a Delay-Constrained Steiner Tree (DCST), which is also NP-complete [6]. An early survey on protocol functions and mechanisms for data transmission within a group and related solutions was given in [7]. A recent overview has been presented in [8] on applications of combinatorial optimization problems and associated algorithms for multicast routing.

In this paper, we investigate a GRASP approach with VNS as the local search method for DCLC multicast routing problems. To our knowledge, very little attention has been given to

Ying Xu

The Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science,
University of Nottingham, Nottingham, NG8 1BB, UK
School of Computer and Communication, Hunan University, Changsha, 410082, CHINA
E-mail: yxx@cs.nott.ac.uk

Rong Qu

The Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science,
University of Nottingham, Nottingham, NG8 1BB, UK
E-mail: rxq@cs.nott.ac.uk

the GRASP approach on multicast routing and we know only one exception in [9]. We tested our proposed GRASP-VNS algorithm on a set of benchmark problems (*steinb* [10]) for Steiner tree problems in the OR library. Computational results indicate that the GRASP-VNS algorithm obtains the same or better quality solutions compared with other three algorithms: *Multi-VNS* (a multi-start algorithm of the extension of our previous variable neighborhood search algorithm [11]), the GRASP algorithm in [9] and *VNSMR2* [11]. Furthermore, we tested our GRASP-VNS algorithm on a set of random graphs. Our proposed GRASP-VNS algorithm performs the best in terms of the total tree cost in comparison with the existing algorithms and heuristics.

The rest of the paper is organized as follows. In section 2, we present the problem definition and related work. Section 3 presents the proposed GRASP-VNS algorithm. We evaluate our algorithm by computer simulations on a range of problem instances and summarize the obtained simulation results in section 4. Finally, section 5 concludes this paper and presents the possible future work.

2 The problem definition and related work

2.1 The network model and problem definition

A computer network can be modeled as a connected, directed graph $G = (V, E)$ with $|V| = n$ nodes and $|E| = l$ links, where V is a set of nodes and E is a set of links. Each link $e = (i, j) \in E$ is associated with two real value functions, namely link cost $c(e)$ and link delay $d(e)$. The link cost $c(e)$ is a measure of the utilization of the corresponding link's resources. The link delay $d(e)$ is the delay caused by transferring messages along the link. Due to the asymmetric nature of computer networks, for link $e = (i, j)$ and link $e' = (j, i)$, it is possible that $c(e) \neq c(e')$ and $d(e) \neq d(e')$. The nodes in V include a source node s and a set of destination nodes D called multicast groups, which receive data stream from the source, denoted by $D \subseteq V \setminus \{s\}$.

We define a path from node u to v as an ordered set of links, denoted by $P(u, v) = \{(u, i), (i, j), \dots, (k, v)\}$. A multicast tree $T(s, D) \subseteq E$ is a tree rooted at source s and spanning all destination nodes in D . We denote by $P(s, r_i) \subseteq T$ the set of links in T that constitute the path from s to $r_i \in D$. The end-to-end delay from s to each destination r_i , denoted by $Delay(r_i)$, is the sum of the delays of all links along $P(s, r_i)$, i.e.

$$Delay(r_i) = \sum_{e \in P(s, r_i)} d(e), \quad \forall r_i \in D \quad (1)$$

The delay of the tree, denoted by $Delay(T)$, is the maximum delay among all $Delay(r_i)$ from source to each destination, i.e.

$$Delay(T) = \max\{ Delay(r_i) \mid \forall r_i \in D \} \quad (2)$$

The total cost of the tree, denoted by $Cost(T)$, is defined as the sum of the cost of all links in the tree, i.e.

$$Cost(T) = \sum_{e \in T} c(e) \quad (3)$$

The delay bound is the upper bound for each $Delay(r_i)$ along the path from s to r_i . Applications may assign different upper bound δ_i to each destination $r_i \in D$. In this paper, we assume that the upper bounds for all destinations are the same, and is denoted by $\Delta = \delta_i$.

Given these definitions, we formally define the Delay-Constrained Steiner Tree (DCST) problem as:

The Delay-Constrained Steiner Tree (DCST) Problem: Given a network G , a source node s , a destination node set D , a link cost function $c(\cdot)$, a link delay function $d(\cdot)$, and a delay bound Δ , the objective of the DCST Problem is to construct a multicast tree $T(s, D)$ such that

the delay bound is satisfied, and the tree cost $Cost(T)$ is minimized. We define the objective function as:

$$\min\{ Cost(T) \mid P(s, r_i) \subseteq T(s, D), Delay(r_i) \leq \Delta, \forall r_i \in D \} \quad (4)$$

2.2 Related work

The DCST problem has received extensive studies, and consequently many exact and heuristic algorithms have been developed since the first DCLC multicast routing algorithm KPP [12] was presented in 1993. Most of these algorithms can be classified as source-based or destination-based multicast routing algorithms. In source-based algorithms, each node has all the necessary information to construct the multicast tree [12-16]. While destination-based algorithms do not require that each node maintains the entire network status information, and multiple nodes participate in constructing the multicast tree [6,17-19].

In recent years, metaheuristic algorithms such as simulated annealing [20,21], genetic algorithm [22,23], tabu search [24-27], GRASP [9], path relinking [28] and VNS [11] have been investigated for various multicast routing problems. Although GRASP algorithms [29,30] have been applied to solve the Minimum Steiner Tree Problem in Graphs (MSTG), along with many other optimization problems, little attention has been given to GRASP for solving the multicast QoS routing problem. Martins et al. [29] describe a hybrid GRASP heuristic with two local search strategies for the Steiner problem. The proposed algorithms were tested on a group of parallel processors. The computing results show that their GRASP heuristic has high possibilities of finding the optimal solutions. Ribeiro et al. [30] present a hybrid GRASP with weight perturbations and two adaptive path-relinking heuristics on a set of elite solutions for the Steiner problem in graphs. One is the path relinking with complementary move; the other is the path relinking with weight penalization. Experiment results on a broad set of benchmark problems illustrate the effectiveness and the robustness of their GRASP algorithm. Both [29] and [30] are restricted to deal with Steiner tree problems with no constraints. In [9], a GRASP heuristic is developed for delay-constrained multicast routing problem. In the GRASP construction phase, a randomized feasible solution is constructed by Dijkstra's shortest path algorithm. The tabu search heuristic [25] is applied in the local search phase. The best found solution after a given number of iterations is accepted as the final solution. Experiment results show that their algorithm outperforms the KPP [12] and the tabu search algorithm [25] on the same problems. A variable neighborhood descent search algorithm *VNSMR2* [11] is proposed in our previous work for the DCLC multicast routing problem. It employs three neighborhood structures, one is node-based and the other two are based on a path replacement strategy. The three neighborhoods are designed to reduce the tree cost and at the same time satisfy the delay constraint.

3 The GRASP-VNS algorithm

GRASP (Greedy Randomized Adaptive Search Procedure) is an efficient multi-start metaheuristic for a wide range of optimization problems, where each iteration consists of two phases: a construction phase and a local search phase [32]. After creating a feasible solution in the construction phase, a local search is applied to explore the neighborhood of the feasible solution until a local minimum is found. The construction phase builds the feasible solution in a greedy randomized manner by iteratively creating a candidate list of elements, called the restricted candidate list (RCL), by evaluating the elements not yet included in the unfinished solution with a certain greedy function. Elements in RCL can be randomly selected and added in the unfinished solution until a feasible solution is obtained. The size of RCL is limited either by the number of the elements or by the quality of the elements with respect to the best candidate element. To further improve the feasible solution generated in the construction phase, a local search is applied to search for better neighboring solutions of the feasible solutions. Different local search strategies can be used by employing the designed

neighborhood structures. After a given number of iterations, the best overall solution is kept as the final solution. More detailed descriptions of the GRASP heuristic can be found in [31,32].

The construction phase of GRASP makes the search diversified, while the local search phase in the GRASP intensifies the search by exploring the neighborhood of the current solution. GRASP has been successfully applied to a wide range of combinatorial optimization problems [33-35]. An especially appealing characteristic of GRASP is that it is easy to implement and few parameters need to be set and tuned. Our motivation is to apply GRASP in conjunction with our previous VNSMR algorithm for the multicast routing problem. Fig.1 illustrates the pseudo code of our proposed GRASP-VNS algorithm.

```

GRASP-VNS( $G=(V, E), s, D, \Delta, Iter, \alpha$ )
{ //  $s$ : source node;  $D$ : destination set;  $\Delta \geq 0$ : the delay bound
  //  $Iter$ : the number of iterations;  $\alpha$ : parameter for creating greedy randomized solution
  if  $Delay(r_i)$  of the Dijkstra's least delay path  $P(s, r_i) > \Delta, \forall r_i \in D$ ;
  then return FAILED; // no feasible solution exists
  else
  {
     $i = 0$ ;
    while  $i < Iter$  do {
      if  $i == 0$  // pure greedy solution ( $\alpha = 1$ ) in the first iteration
         $T_{best} = GreedyRandomSolution(G, s, D, \Delta, 1)$ ; //the construction phase, see Fig. 2
      else // the remaining iterations of GRASP-VNS ( $\alpha > 1$ )
         $T_0 = GreedyRandomSolution(G, s, D, \Delta, \alpha)$ ; //the construction phase, see Fig. 2
         $T = VNSMR2(G, s, D, \Delta, T_0)$ ; // the local search phase
        if  $((Cost(T) < Cost(T_{best})) \vee ((Cost(T) == Cost(T_{best})) \wedge (Delay(T) < Delay(T_{best}))))$ 
        then  $T_{best} = T$ ;
         $i++$ ;
      } //end of while loop
    }
    return  $T_{best}$ ;
  }
}

```

Fig. 1. The pseudo code of the GRASP-VNS algorithm

3.1 The construction phase

In the construction phase of our GRASP-VNS algorithm, we use the similar greedy strategy in [9] to create the randomized initial solution. A delay-constrained Steiner tree T is constructed in the following three steps. The pseudo code of the construction phase is given in Fig.2.

- (1) Starting from the source node, i.e. $T = \{s\}$, we calculate the shortest path which connects s and each destination by using the Dijkstra's shortest path algorithm. This path is denoted as $ConnectPath[r_i]$, for each destination node $r_i \in D \setminus T$. We set $ConnectPath[r_i]$ as the least cost path from s to r_i if the delay of the path satisfies the delay bound; otherwise, $ConnectPath[r_i]$ is set as the least delay path from s to r_i . We denote $ConnectCost[r_i]$ as the cost of the path.
- (2) We create the restricted candidate list (RCL) by choosing those nodes $r_i \in D \setminus T$, for which $ConnectCost[r_i] \leq \alpha \cdot BestConnectCost$, where $\alpha \geq 1$ and $BestConnectCost = \min\{ConnectCost[r_i], \forall r_i \in D \setminus T\}$. If $\alpha = 1$, then the algorithm is purely greedy. This means the RCL contains only the destination node with the least connect cost. If $\alpha > 1$, the RCL includes the nodes with the path cost within the range of $\alpha \cdot BestConnectCost$.
- (3) We randomly choose a destination node r from the RCL and add $ConnectPath[r]$ to the tree T . We update the $ConnectPath[r_i]$ and $ConnectCost[r_i]$ of the remaining unconnected destination nodes in $D \setminus T$ by searching the new shortest path from these destination nodes to the current tree T . After that, the algorithm will return to step (2). The procedure ends until all the destination nodes are included in the tree.

In order to construct a high quality starting solution, we use the pure greedy algorithm in the first iteration i.e. $\alpha = 1$. In the remaining iterations, we set $\alpha > 1$ which gives more diversity to the construction phase to explore the solution space.

3.2 The local search phase

After a feasible solution is generated in the construction phase, we apply our *VNSMR2* developed in [11] as the local search method to improve the initial solution. It systematically changes the employment of different neighborhoods within a local search, thus the search is more flexible to traverse among different search spaces and potentially leads to better solutions. Three neighborhood structures are designed for multicast routing problems. The first neighborhood based on the nodes operation generates a neighboring solution by deleting a node from the current solution and then using Prim's algorithm to create the minimum spanning tree of the remaining nodes. The other two neighborhoods operate on the paths by using a path replacement strategy based on the Dijkstra's shortest path algorithm to reduce the tree cost of the current solution and at the same time satisfy the delay bound. *VNSMR2* was implemented based on the variable neighborhood decent search algorithm, a variant of variable neighborhood search (VNS) [36], where the current solution is always updated by the best neighboring solution in each neighborhood structure. In this paper we hybridize *VNSMR2* within the GRASP approach for solving not only random multicast routing problems but also more challenging benchmark problems in the OR Library.

```

GreedyRandomSolution( $G=(V, E), s, D, \Delta, \alpha$ )
{
  // s: source node; D: destination set;  $\Delta \geq 0$  is the delay bound
  //  $\alpha$  is parameter to create the RCL (restricted candidate list)
  for all  $r_i \in D$ 
    Calculate ConnectPath[ $r_i$ ] and ConnectCost[ $r_i$ ];
   $T = s$ ;
  while  $D \not\subset T$  do {
     $BestConnectCost = \min\{ConnectCost[r_i], \forall r_i \in D \setminus T\}$ ;
    Create RCL of all  $r_i \in D \setminus T$  where  $ConnectCost[r_i] \leq \alpha \cdot BestConnectCost$ ;
     $r = \text{Random}(RCL, Randseed)$ ; //Randomly choose  $r$  from RCL
                                //  $Randseed \in (0,1]$ 
     $T = T \cup ConnectPath[r]$ ;
    Update ConnectPath[ $r_j$ ] and ConnectCost[ $r_j$ ] for all  $r_j \in D \setminus T$ ;
  } //end of while loop
  return  $T$ ;
}

```

Fig. 2. The pseudo code of the construction phase of *GRASP-VNS*

4 Performance Evaluations

We use a multicast routing simulator (MRSIM) implemented in C++ based on Salama's generator [1] to generate random network topologies. The simulator defines the link delay function $d(e)$ as the propagation delay of the link (queuing and transmission delays are negligible). The link cost function $c(e)$ is defined as the current total bandwidth reserved on the link in the network, and is related to the distance of the link. The Euclidean metric is used to determine the distance $l(u, v)$ between pairs of nodes (u, v) . Links connecting nodes (u, v) are placed with a probability

$$p(u, v) = \beta \exp(-l(u, v) / \alpha L) \quad \alpha, \beta \in (0, 1] \quad (5)$$

where parameters α and β can be set to obtain desired characteristics in the graph. For example, a large β gives nodes a high average degree, and a small α gives long connections in the networks. L is the maximum distance between two nodes. In our simulations, we set $\alpha = 0.25$, $\beta = 0.40$, average degree = 4. All simulations were run on a Windows XP computer with PVI 3.4GHZ, 1G RAM. More detailed information of the test datasets and some example

solutions obtained by the algorithms on the tested instances are publicly available at <http://www.cs.nott.ac.uk/~yxx/resource.html>.

4.1 Experiments on benchmark problems (*steinb*) in OR-library

Firstly, experiments were carried out on 18 small and medium sized (50-100 nodes) *steinb* benchmark problems in the OR library, details given in Table 1. Since *steinb* problems are for the Steiner tree problem, only a cost function is assigned to the links in each benchmark problem. In our experiments, we randomly set the delays of the links when generating the network topology in the simulator for each *steinb* problem. The simulation was run 10 times on each instance. We implemented our proposed *GRASP-VNS* and re-implemented the *GRASP-CST* algorithm in [9]. We also extended *VNSMR2* to a multi-start algorithm by running it for a fixed number of iterations, named *Multi-VNS*. Each iteration of *Multi-VNS* starts from a greedy random initial solution, which is generated by randomly choosing a starting destination node and connecting it with the source by using the Dijkstra's shortest path algorithm, then repeatedly connecting the unvisited destination nodes with the sub-tree until all the destination nodes have been mounted to the tree.

Table 1. The problem characteristics of dataset *steinb* from the OR-library ($|V|$, $|E|$, $|D|$ denote the number of nodes, the number of edges and the number of destinations in the instances respectively, 'OPT' denotes the optimal solution)

No.	$ V $	$ E $	$ D $	OPT	No.	$ V $	$ E $	$ D $	OPT	No.	$ V $	$ E $	$ D $	OPT
B01	50	63	9	82	B07	75	94	13	111	B13	100	125	17	165
B02	50	63	13	83	B08	75	94	19	104	B14	100	125	25	235
B03	50	63	25	138	B09	75	94	38	220	B15	100	50	125	318
B04	50	100	9	59	B10	75	150	13	86	B16	100	17	200	127
B05	50	100	13	61	B11	75	150	19	88	B17	100	25	200	131
B06	50	100	25	122	B12	75	150	38	174	B18	100	50	200	218

To determine appropriate settings for the parameters in our *GRASP-VNS* algorithm, a number of initial tests were carried out. Our aim was to obtain good quality solutions by using as less number of iterations as possible to reduce the execution time. Parameter α (see Fig. 2.) should be properly set to find the balance between diversification and intensification of the search over the search space. In other words, we should find a trade off between the solution quality and the execution time. After the initial tests, the number of iterations is set as 4, α is set as 5 in the *GRASP-VNS* algorithm. Parameters for *GRASP-CST* are set as the same as in [9], where the number of iterations is 5, α is 5, and the number of iterations without improvement of the local search procedure is 2. The number of iterations is set to 4 in *Multi-VNS*.

In the first group of experiments, we set the delay bound to a large enough number so that the problems are reduced to the unconstrained Steiner tree problem since the delays of the links play no role in constructing the Steiner tree. The average, best, worst tree cost and the computing time of *GRASP-VNS*, compared with that of *Multi-VNS*, *GRASP-CST* and *VNSMR2*, are illustrated in Table 2. We can see that the *GRASP-VNS* algorithm gave the best solutions (marked in bold) on all the instances except one (B13) with respect to the average tree cost, while *Multi-VNS* algorithm got 14 best solutions, both *GRASP-CST* and *VNSMR2* found 10 best results. The *GRASP-VNS* algorithm always found the optimal solutions in 14 out of 18 instances, which is better than *Multi-VNS*, *GRASP-CST* and *VNSMR2* which always got the optimal solutions on 13, 10 and 10 instances, respectively. Both *GRASP-VNS* and *GRASP-CST* found the optimal solution at least once out of 10 runs for each instance. It should be noticed that the *GRASP-VNS* algorithm achieved better results by consuming longer computing time than *GRASP-CST*.

For the DCST multicast routing problem, the delay bound is the key factor which affects the solutions obtained. The smaller the delay bound, the stronger the constraint. In the second

group of experiments, we set the delay bound $\Delta_1 = 1.1 * Delay(T_{OPT})$, where T_{OPT} is the multicast tree of the optimal solution with the minimal cost and delay. With the bounded end-to-end delay, the *GRASP-VNS* algorithm still performs the best among these three algorithms in terms of average tree costs in Table 3. *GRASP-VNS* found the best solutions in 15 out of 18 instances, compared with that of *Multi-VNS* (13 best solutions), *GRASP-CST* and *VNSMR2* (9 best solutions). *GRASP-VNS* always found the optimal solutions in 12 out of 18 cases, while *Multi-VNS*, *GRASP-CST* and *VNSMR2* did so in 11, 8 and 10 cases, respectively.

Table 2. Performance of *GRASP-VNS*, *Multi-VNS*, *GRASP-CST* and *VNSMR2* for unconstrained Steiner tree problems ($\Delta = \infty$). The values marked with ‘*’ denote the optimal solution.

No.	<i>GRASP-VNS</i>				<i>Multi-VNS</i>				<i>GRASP-CST</i>				<i>VNSMR2</i>			
	Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)
B01	82*	82*	82*	0.79	82*	82*	82*	0.49	82*	82*	82*	0.63	82*	82*	82*	0.13
B02	83*	83*	83*	1.46	83*	83*	83*	1.95	83*	83*	83*	0.80	83*	83*	83*	0.18
B03	138*	138*	138*	2.12	138*	138*	138*	2.09	138*	138*	138*	1.10	138*	138*	138*	0.26
B04	59*	59*	59*	1.41	59*	59*	59*	1.74	59*	59*	59*	0.68	59*	59*	59*	0.15
B05	61*	61*	61*	2.11	61*	61*	61*	2.63	61*	61*	61*	0.86	61*	61*	61*	0.22
B06	122.8	122*	124	3.49	124	124	124	4.11	123.1	122*	125	1.80	125	125	125	0.41
B07	111*	111*	111*	3.39	111*	111*	192	3.81	111*	111*	111*	1.86	111*	111*	111*	0.26
B08	104*	104*	104*	6.11	105.2	104*	107	5.02	104*	104*	104*	2.80	107	107	107	0.55
B09	220*	220*	220*	9.38	220*	220*	220*	8.68	220*	220*	220*	4.92	220*	220*	220*	1.11
B10	86*	86*	86*	4.9	86*	86*	86*	7.39	86.5	86*	91	2.39	88.5	86*	91	0.73
B11	88*	88*	88*	6.31	88*	88*	88*	10.12	88.1	88*	89	2.93	89.6	88*	90	0.79
B12	174*	174*	174*	14.86	174*	174*	174*	14.56	174*	174*	174*	6.76	174*	174*	174*	1.75
B13	168.6	165*	172	14.06	166.6	165*	169	18.96	169.7	165*	173	6.98	172	172	172	2.23
B14	235.2	235*	236	21.01	236	236	236	47.4	235.4	235*	236	8.57	236	236	236	1.78
B15	319.8	318*	320	37.39	320	320	320	41.21	321.2	318*	322	16.56	321	321	321	2.94
B16	127*	127*	127*	16.9	127*	127*	127*	26.38	128	127*	132	6.41	127*	127*	127*	3.06
B17	131*	131*	131*	28.93	131*	131*	131*	24.79	131*	131*	131*	10.3	131*	131*	131*	4.44
B18	218*	218*	218*	41.12	218*	218*	218*	34.41	218.4	218*	219	19.28	218.1	218*	219	3.76

Table 3. Performance of *GRASP-VNS*, *Multi-VNS*, *GRASP-CST* and *VNSMR2* for Steiner tree problems with $\Delta_1 = 1.1 * Delay(T_{OPT})$

No.	Δ	<i>GRASP-VNS</i>				<i>Multi-VNS</i>				<i>GRASP-CST</i>				<i>VNSMR2</i>			
		Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)
B01	145	82*	82*	82*	0.48	82*	82*	82*	0.46	82*	82*	82*	0.54	82*	82*	82*	0.07
B02	228	83*	83*	83*	1.45	83*	83*	83*	1.95	83*	83*	83*	0.8	83*	83*	83*	0.18
B03	248	138*	138*	138*	2.14	138*	138*	138*	2.05	138*	138*	138*	1.09	138*	138*	138*	0.26
B04	173	59*	59*	59*	1.23	59*	59*	59*	1.71	59*	59*	59*	0.64	59*	59*	59*	0.14
B05	125	61*	61*	61*	2.01	61*	61*	61*	2.64	61*	61*	61*	0.81	61*	61*	61*	0.21
B06	281	122.2	122*	124	3.58	124	124	124	3.9	123	122*	124	1.73	125	125	125	0.41
B07	212	111*	111*	111*	3.36	111*	111*	111*	3.77	111*	111*	111*	1.83	111*	111*	111*	0.26
B08	209	104*	104*	104*	6.35	104.6	104*	107	5.44	104*	104*	104*	2.8	107	107	107	0.55
B09	280	220*	220*	220*	9.33	220*	220*	220*	8.6	220*	220*	220*	4.87	220*	220*	220*	1.11
B10	262	86.5	86*	91	4.91	86*	86*	86*	7.51	87	86*	91	2.20	88.5	86*	91	0.73
B11	235	88*	88*	88*	6.7	88.2	88*	90	9.74	88.3	88*	90	2.92	89.7	88*	91	0.9
B12	225	174*	174*	174*	15.87	174*	174*	174*	12.52	175	174*	178	6.37	174*	174*	174*	2.07
B13	190	169.4	165*	172	15.48	167	165*	169	19.05	170.6	165*	173	6.67	172	172	172	2.23
B14	221	235*	235*	235*	15.5	244.1	236	245	14.28	235.2	235*	236	8.19	236	236	236	1.81
B15	308	319.5	318*	320	36.22	320	320	320	38.76	319.8	318*	320	17.94	321	321	321	5.27
B16	291	127*	127*	127*	17.06	127*	127*	127*	24.93	130	127*	132	6.12	127*	127*	127*	3.07
B17	219	131.5	131*	132	25.13	131*	131*	131*	24.21	131.9	131*	132	9.64	132	132	132	4.46
B18	425	218.1	218*	219	40.32	218.1	218*	219	34.37	218.1	218*	219	19.54	218.4	218*	219	3.77

In the third group of experiments, we set the delay bound Δ_2 to a smaller value $Delay(T_{OPT})$, thus the optimal solutions are not known to any of the cases. Table 4 shows that again *GRASP-VNS* outperforms *Multi-VNS* and *GRASP-CST* upon the average tree costs on 13 instances. *GRASP-VNS*, *GRASP-CST* and *VNSMR2* could not find the feasible solutions on 2 instances (B03, B07) due to the tighter delay bound. The *Multi-VNS* also failed to find the

feasible solutions on instance B07 and B14. The *GRASP-CST*, *Multi-VNS*, *GRASP-CST* and *VNSMR2* obtained 13, 8, 5 and 5 best solutions out of 16 instances, respectively. With regards to the average computing time for the instances, *Multi-VNS* requires longer time (13.872 seconds) than *GRASP-VNS* (11.803 seconds), *GRASP-CST* (5.333 seconds) and *VNSMR2* (1.464 seconds).

Table 4. Performance of *GRASP-VNS*, *Multi-VNS*, *GRASP-CST* and *VNSMR2* for Steiner tree problems with $\Delta_2 = 0.9 * Delay(T_{OPT})$. “\” denotes that no feasible solution was obtained.

No.	Δ	<i>GRASP-VNS</i>				<i>Multi-VNS</i>				<i>GRASP-CST</i>				<i>VNSMR2</i>			
		Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)	Avg.	Best	Worst	Time(s)
B01	118	83	83	83	0.42	83	83	83	0.31	83	83	83	0.46	83	83	83	0.06
B02	187	84	84	84	0.77	84	84	84	1.43	84	84	84	0.63	84	84	84	0.09
B03	203	\	\	\	\	142	142	142	1.81	\	\	\	\	\	\	\	\
B04	142	62.4	62	64	1.26	63.8	62	64	2.75	62.6	62	65	0.65	66.2	62	76	0.2
B05	102	62	62	62	1.52	62.1	62	63	2.26	64.7	62	65	0.54	62	62	62	0.26
B06	199	123.9	123	124	3.81	124.3	124	125	4.8	124	124	124	1.58	129	129	129	0.4
B07	173	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\
B08	171	107	107	107	5.75	107	107	107	5.19	107	107	107	2.59	107	107	107	0.55
B09	229	221	221	221	8.68	221	221	221	9.15	221	221	221	5.05	221	221	221	1.05
B10	215	88.3	88	91	5.08	88	88	88	7.72	88.3	88	91	2.12	89.2	88	91	0.8
B11	180	89.2	89	91	6.48	91.5	89	93	9.75	89.2	89	90	3.09	91	90	93	1.05
B12	184	176.8	176	177	12.95	177.2	177	179	16.41	178.5	177	189	6.37	197.6	177	202	1.73
B13	139	171.2	169	172	15.72	169	169	169	19.99	172	168	173	6.62	173	173	173	1.38
B14	180	237.2	237	239	16.66	\	\	\	\	242.1	237	243	6.83	239	239	239	3.49
B15	194	327.1	326	329	38.84	328.7	326	329	40.27	330.1	328	331	15.62	341.6	339	342	7.27
B16	238	131.1	129	132	12.84	132	132	132	26.99	131.7	129	132	5.46	132	132	132	1.8
B17	180	134.2	134	135	20.98	134	134	134	33.53	134.8	134	135	8.34	135	135	135	4.36
B18	348	219.2	219	220	38.93	219.8	219	220	39.59	219.6	219	220	18.55	219.5	219	220	3.73

Further experiments are designed to test how the algorithms evolve within a given period of time. Examples of the evolution process of these algorithms on one instance (B15) in the above *steinb* problems are shown in Fig.3. Here, we set the delay bound to a very large number so as not to act as a constraint. In Fig.3.(a), we can see that our *GRASP-VNS* algorithm converged faster than *GRASP-CST* and *Multi-VNSMR2* in the given 10 iterations. The *GRASP-VNS* algorithm finds the optimal solution at iteration 3, *GRASP-CST* finds the optimal solution at iteration 8, while *Multi-VNS* does not find the optimum within the 10 iterations. Furthermore, the three algorithms were tested by giving the exact same amount of time (60 seconds). Fig.3.(b) shows that our *GRASP-VNS* still converged faster than *GRASP-CST* and *Multi-VNS* on the instance B15.

All experiment results demonstrate that the *GRASP-VNS* algorithm has the overall best performance compared with *Multi-VNS*, *GRASP-CST* and *VNSMR2* in terms of the average tree cost for this set of benchmark problems. The proposed *GRASP-VNS* algorithm, which applies GRASP metaheuristic by building the RCL in the construction phase, outperforms the simple multi-start *Multi-VNS* algorithm with only randomized initial solutions in terms of both average tree cost and computing time in most cases. Both *GRASP-VNS* and *Multi-VNS*, the extension of our previous *VNSMR2* algorithm, outperform *VNSMR2*, indicating the two multi-start algorithms are more robust and efficient than the single *VNSMR2* algorithm with only one variable neighborhood search phase when exploring the solution space of hard problems.

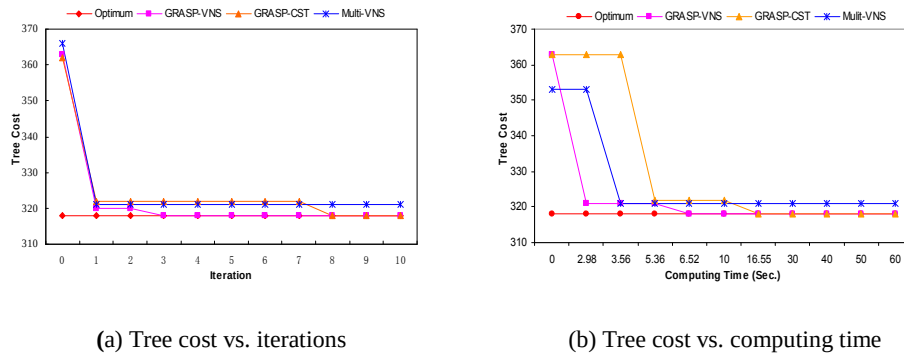


Fig. 3. Evolution process of the three algorithms on instance B15

4.2 Experiments on random graphs

In order to compare our *GRASP-VNS* algorithm with other existing algorithms, we tested *GRASP-VNS* on a group of randomly generated graphs which are the same simulation data tested in our previous work [11]. These random graphs include three random topologies for each network size (10, 20, 30, 40, 50, 60, 70, 80, 90, 100 nodes). The link costs are set depending on the length of the link; all the link delays are set to 1. The group size was set to 30% of the network size in each graph, the delay bounds were set to different values depending on the network sizes ($\Delta = 7$ for network size 10-30, $\Delta = 8$ for network size 40-60, $\Delta = 10$ for network size 70-80 and $\Delta = 12$ for network size 90-100). The simulation was run 10 times on each random graph.

Table 5. Average tree costs of our *GRASP-VNS* and some existing heuristics and algorithms on the random graphs

Algorithms		Average Tree Cost
Heuristics	KPP1 [12]	905.581
	KPP2 [12]	911.684
	BSMA [16]	872.681
GA-based Algorithms	Wang et al. [22]	815.969
	Haghighat et al. [23]	808.406
TS-based Algorithms	Skorin-Kapov and Kos [25]	897.875
	Youssef et al. [24]	854.839
	Wang et al. [26]	869.291
	Ghaboosi and Haghighat [27]	739.095
Path relinking	Ghaboosi and Haghighat [28]	691.434
VNS Algorithms	VNSMR1 [11]	680.067
	VNSMR2 [11]	658.967
	Multi-VNS	653.257
GRASP Algorithms	GRASP-CST [9]	669.927
	Our proposed GRASP-VNS algorithm	649.203

Table 5 shows that our *GRASP-VNS* algorithm performs the best in terms of the average tree cost. Both *Multi-VNS* and *VNSMR2* get better average tree costs than *GRASP-CST*. Details of the average tree cost, standard deviation and execution time of these three algorithms on each network size are given in Table 6. It shows that *GRASP-VNS* found the best solutions in 8 out of 10 network sizes, while *GRASP-CST* and *Multi-VNS* obtained the best results twice on the 10 types of random graphs. With respect to the average standard deviation σ , *GRASP-VNS* has lower σ (5.316) than *GRASP-CST* (7.238), but higher σ than *Multi-VNS* (4.641). To summarize, our *GRASP-VNS* algorithm gives high quality solutions and stable on most of the tested random graphs. *GRASP-VNS* has longer average computing time (11.006 seconds) in

comparison with *GRASP-CST* (3.289 seconds), however it is better than *Multi-VNS* which has the longest computing time (12.737 seconds).

Table 6. Average tree cost, standard deviation of the tree cost and execution time of *GRASP-VNS*, *Multi-VNS*, *GRASP-CST* and *VNSMR2* on the random graphs

Network Size	<i>GRASP-VNS</i>			<i>Multi-VNS</i>			<i>GRASP-CST</i>			<i>VNSMR2</i>		
	Cost	σ	Time(s)	Cost	σ	Time(s)	Cost	σ	Time(s)	Cost	σ	Time(s)
10	94.67	0	0.008	94.67	0	0.008	94.67	0	0.009	94.67	0	0.003
20	272.07	2.25	0.085	275.33	0	0.089	271.13	1.48	0.048	275.33	0	0.032
30	392.33	0	0.353	395.27	3.95	0.461	394.67	0	0.156	399.67	0	0.17
40	512.8	1.55	0.857	513.33	0	1.238	526.47	1.79	0.388	514	0	0.362
50	662.33	1.94	2.109	665.07	3.92	3.027	697.07	3.43	0.815	674.67	0	0.859
60	757.33	13.48	3.894	757.37	14.01	4.894	761.13	17.13	1.625	777.67	0	1.392
70	780.83	2.96	9.029	796.2	4.48	9.268	797.53	1.64	2.648	805	0	2.571
80	868.87	7.73	19.421	896.2	4.48	16.365	902.67	5.49	5.941	905.33	0	5.127
90	1155.57	19.02	32.621	1136.23	11.17	38.76	1201.93	18.02	10.27	1137.67	0	11.705
100	995.23	4.23	41.681	1002.9	4.4	53.256	1052	23.4	10.983	1005.67	0	15.332

5. Conclusions

In this paper, we have investigated a *GRASP-VNS* algorithm (Greedy Randomized Adaptive Search Procedure approach with Variable Neighborhood Search) for solving Delay-Constrained Least-Cost multicast routing problems. The problem is a special case of Delay-Constrained Steiner tree (DCST) problem and has been proved to be NP-complete. Although *GRASP* is an efficient metaheuristic for optimization problems, little attention has been given on applying it for solving the QoS constrained multicast routing problem. A large number of experiments have been carried out on a set of benchmark problems for Steiner tree problems in the OR-library and a group of random graphs. Experiment results show that the proposed algorithm performs the best in comparison with some existing algorithms for all the tested instances in terms of average tree cost. Experiments demonstrate that our *GRASP-VNS* algorithm is able to find high quality solutions for DCST multicast routing problems and efficiently solve benchmark Steiner tree problems.

Many interesting future research directions could be explored. The introduction of multiple QoS constraints, such as the bandwidth, delay-variation or node degrees to the multicast routing problem deserves further investigation. In reality, network scenarios are mostly dynamic with multicast members leaving and joining the multicast group at various times. The adaptation and extension of *GRASP* approaches to the problem of dynamic multicast routing is worthy of further investigation.

Acknowledgements This research is supported by Hunan University, China, and the School of Computer Science at The University of Nottingham, UK.

References

1. Salama H.F., Reeves D.S., Viniotis Y., Evaluation of multicast routing algorithms for real-time communication on high-speed networks, *IEEE Journal on Selected Areas in Communications*, 15(3), 332--345 (1997)
2. Yeo C.K., Lee B.S., Er M.H., A survey of application level multicast techniques, *Computer Communications*, 27(15), 1547--1568 (2004)
3. Masip-Bruin X., Yannuzzi M., Domingo-Pascual J., Fonte A., Curado M., Monteiro E., Kuipers F., Van Mieghem P., Avallone S., Ventre G., Aranda-Gutierrez P., Hollick M., Steinmetz R., Iannone L., Salamatin K., Research challenges in QoS routing, *Computer Communications*, 29(5), 563--581 (2006)

4. Cheng X., Du D.Z., (eds.) *Steiner Trees in Industry*, Kluwer Academic Publishers, Dordrecht, Netherlands (2001)
5. Garey M.R., Johnson D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York (1979)
6. Guo L., Matta I., QDMR: An efficient QoS dependent multicast routing algorithm. In: *Proceedings of the 5th IEEE RealTime Technology and Applications Symposium*, pp. 213--222 (1999)
7. Diot C., Dabbous W., Crowcroft J., Multipoint communication: a survey of protocols, functions, and mechanisms, *IEEE Journal on Selected Areas in Communications*, 15(3), 277--290 (1997)
8. Oliveira C.A.S., Pardalos P.M., A survey of combinatorial optimization problems in multicast routing. *Computers & Operations Research*, 32(8), 1953--1981 (2005)
9. Skorin-Kapov N., Kos M., A GRASP heuristic for the delay-constrained multicast routing problem, *Telecommunication Systems*, 32(1), 55--69 (2006)
10. <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/steininfo.html>
11. Qu R., Xu Y., Kendall G., A Variable Neighborhood Descent Search Algorithm for Delay-Constrained Least-Cost Multicast Routing, In: *Proceedings of Learning and Intelligent OptimizatioN (LION3)*, Trento, Italy (2009)
12. Kompella V.P., Pasquale J.C., Polyzos G.C., Multicast routing for multimedia communication, *IEEE/ACM Transactions on Networking*, 1, 286--292 (1993)
13. Widyno R., The design and evaluation of routing algorithms for realtime channels. Technical Report, ICSI TR-94-024, International Computer Science Institute, U.C. Berkeley (1994)
14. Sun Q., Langendoerfer H., An efficient delay-constrained multicast routing algorithm. Technical Report, Internal Report, Institute of Operating Systems and Computer Networks, TU Braunschweig, Germany (1997)
15. Sun Q., Langendoerfer H., Efficient multicast routing for delay-sensitive applications, In: *Proceedings of the 2nd workshop on protocols for multimedia systems*, pp. 452--458 (1995)
16. Zhu Q., Parsa M., Garcia-Luna-Aceves J. J., A source-based algorithm for delay-constrained minimum-cost multicasting, In: *Proceedings of the 14th Annual Joint Conference of the IEEE Computer and Communication (INFOCOM'95)*, pp. 377--385. IEEE Computer Society Press, Washington, DC, USA (1995)
17. Kompella V.P., Pasquale J.C., Polyzos G.C., Two distributed algorithms for the constrained Steiner tree problem, In: *Proceedings of the 2nd International Conference on Computer Communications and Networking*, pp.343--349 (1993)
18. Shaikh A., Shin K., Destination-driven routing for low-cost multicast, *IEEE Journal on Selected Areas in Communications*, 15, 373--381 (1997)
19. Jia X., A distributed algorithm of delay-bounded multicast routing for multimedia applications in wide area networks, *IEEE/ACM Transactions on Networking*, 6, 828--837 (1998)
20. Wang X.L., Jiang Z., QoS multicast routing based on simulated annealing algorithm, In: *Proceedings international and applications*, pp. 511--516 (2004)
21. Zhang K., Wang H., Liu F.Y. Distributed multicast routing for delay and delay variation-bounded Steiner tree using simulated annealing. *Computer Communications* 28(11): 1356-1370 (2005)
22. Wang Z., Shi B., Zhao E., Bandwidth-delay-constrained least-cost multicast routing based on heuristic genetic algorithm, *Computer communications*, 24, 685--692 (2001)
23. Haghighat A.T., Faez K., Dehghan M. Mowlaei A., Ghahremani Y., GA-based heuristic algorithms for bandwidth-delay-constrained least-cost multicast routing, *Computer Communications*, 27, 111--127 (2004)
24. Youssef H., Al-Mulhem, A., Sait S.M., Tahir M.A., QoS-driven multicast tree generation using tabu search, *Computer Communications*, 25(11-12), 1140--1149 (2002)
25. Skorin-Kapov N., Kos M., The application of Steiner trees to delay constrained multicast routing: a tabu search approach, In: *Proceedings of the seventh international Conference on Telecommunications*, Zagreb, Croatia (2003)
26. Wang H., Fang J., Wang H., Sun Y.M., TSDLMRA: an efficient multicast routing algorithm based on tabu search, *Journal of Network and Computer Applications*, 27, 77--90 (2004)
27. Ghaboosi N., Haghighat A.T., A tabu search based algorithm for multicast routing with QoS constraints, In: *9th International Conference on Information Technology*, pp. 18--21 (2006)
28. Ghaboosi N., Haghighat A.T., A path relinking approach for Delay-Constrained Least-Cost Multicast routing problem, In: *19th International Conference on Tools with Artificial Intelligence*, pp. 383--390 (2007)
29. Martins S.L., Resende M.G.C., Ribeiro C.C., Pardalos P.M., A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy, *Journal of Global Optimization*, 17, 267--283 (2000)

30. Ribeiro C.C., Uchoa E., Werneck R.F., A Hybrid GRASP with Perturbations for the Steiner Problems in Graphs, *INFORMS Journal on Computing*, 14(3), 228--246 (2002)
31. Feo T.A., Resende M.G.C., Greedy randomized adaptive search procedures, *Journal of Global Optimization*, 6, 109--133 (1995)
32. Resende M.G.C., Ribeiro C.C., Greedy randomized adaptive search procedures, In: Glover F., Kochenberger G. (Eds.), *Handbook of Metaheuristics*, Kluwer Academic Publishers, Dordrecht. 219--249 (2003)
33. Kontoravdis G., Bard J.F. A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing*, 7, 10--23 (1995)
34. Pardalos P.M., Qian T., Resende M.G.C. A greedy randomized adaptive search procedure for the feedback vertex set problem. *Journal of Combinatorial Optimization*, 2, 399--412 (1999)
35. Resende M.G.C., Pitsoulis L.S., Pardalos P.M. Fortran subroutines for computing approximate solutions of weighted MAX-SAT problems using GRASP. *Discrete Applied Mathematics*, 100, 95--113 (2000)
36. Mladenovic N., Hansen P., Variable neighborhood search, *Computers & Operations Research*, 24, 1097--1100 (1997)

Ant Colony Optimization Approaches for Scheduling Jobs with Incompatible Families on Parallel Batch Machines

Lars Mönch • Christian Almeder

Abstract In this paper, we suggest an Ant Colony System (ACS) to solve a scheduling problem for jobs with incompatible families on parallel batch machines. We are interested in minimizing total weighted tardiness (TWT) of the jobs. Problems of this type have practical importance in semiconductor manufacturing. The ACS scheme includes an efficient local search technique based on swapping jobs across different batches of the same family. A comparison of the suggested ACS scheme with a dispatching rule based approach and a genetic algorithm (GA) from previous research is performed based on stochastically generated test instances. It turns out that the ACS approach slightly outperforms the GA with respect to solution quality but it provides the results using considerably less computational time. We describe extensive computational experimentation to find appropriate parameters for the ACS scheme. Finally, we also present a MAX-MIN Ant System (MMAS) and compare its performance with the ACS approach. MMAS provides basically the same solution quality, but it requires much more computational efforts.

1 Introduction

This research is motivated by a scheduling problem found in the area of diffusion in a semiconductor wafer fabrication facility (wafer fab). In the diffusion area, there are several diffusion furnaces which can be operated in parallel and can process several jobs simultaneously. However, due to the different chemical nature of the processes, only jobs of the same job family can be processed together. Carefully choosing which products to batch together and which order these jobs should be sequenced is of high importance as the processing times in the diffusion furnaces can take up to 10 hours, versus a 1 hour processing time in other wafer fab processes. Therefore, a proper scheduling of the machines dedicated to the diffusion process steps has a great impact on the performance of the entire wafer fab. Given that many wafer fabs are driven by on-time delivery, the performance measure of interest is TWT. This performance measure is given as $\sum w_{js} T_{js}$, where we denote by w_{js} the weight or priority for job j of family s . The tardiness of job j of family s is given by $T_{js} := \max(C_{js} - d_{js}, 0)$, where we denote by C_{js} the completion time for job j of family s in the schedule and d_{js} represents the due date for job j of family s .

In this paper, we extend previous research on parallel batch process machines using genetic algorithms (GA) [1]. We suggest an ACS and a MMAS scheme. In contrast to other ACO approaches for parallel machines we use a pure sequencing representation that includes the assignment of the jobs to batches and machines only indirectly. Our ACO schemes are hybridized with a local search technique that changes the content of the batches across

Lars Mönch
University of Hagen
E-mail: Lars.Moench@fernuni-hagen.de

Christian Almeder
University of Vienna
E-mail: Christian.Almeder@univie.ac.at

machines and also the sequence of batches on each machine. We show that ACS slightly outperforms the GA approach with respect to solution quality. But ACS requires considerably less computing time. The performance of MMAS is similar to ACS but much more computing time is required. This kind of comparison between different ACO schemes for the same scheduling problem is rarely discussed in the literature.

The paper is organized as follows. In the next section, we describe the problem under consideration in detail, present a Mixed Integer Programming (MIP) formulation, and discuss related literature. Then we describe our ACS scheme and the MMAS approach. We present results of computational experiments with both heuristics in the last section of this paper.

2 Problem Description and Related Literature

2.1 Problem Setting

For the problem being researched in this paper the $\alpha | \beta | \gamma$ notation is:

$$Pm | p\text{-batch, incompatible} | \sum w_j T_j . \quad (1)$$

There are f families of jobs and m parallel machines. Family s has n_s jobs. The total number of jobs is n . All jobs of family s have the same processing time. The jobs that are waiting to be processed must be batched by family, depending on the maximum batch size B . Each single job has a weight w_{js} and a due date d_{js} . Our objective is to minimize the TWT measure. Scheduling problem (1) is NP hard because the scheduling problem $1 || TWT$ is NP hard. Therefore, we have to look for efficient heuristics.

We present a MIP formulation for problem (1) in order to assess the solution quality of the suggested heuristics for small size test instances. We start with introducing the necessary index sets:

$$\begin{aligned} b = 1, \dots, n_b & \quad \text{batch index,} & j = 1, \dots, n & \quad \text{job index,} \\ s = 1, \dots, f & \quad \text{job family index,} & l = 1, \dots, m & \quad \text{machine index.} \end{aligned}$$

Furthermore, we present the necessary parameters:

$$\begin{aligned} B & \quad \text{maximum batch size,} & p_s & \quad \text{processing time of family } s, \\ d_j & \quad \text{due date of job } j, & w_j & \quad \text{weight of job } j, \\ e_{js} & \quad \begin{cases} 1 & \text{if job } j \text{ belongs to family } s \\ 0 & \text{otherwise} \end{cases}, & M & \quad \text{large number.} \end{aligned}$$

We consider four decision variables. The completion time of the b -th batch on machine l is denoted by C_{bl} . The tardiness of job j is denoted by T_j . C_{bl} and T_j are real-valued variables. Furthermore, the following two binary variables are used:

$$\begin{aligned} X_{jbl} &:= \begin{cases} 1 & \text{if job } j \text{ is assigned to the } b\text{-th batch of machine } l \\ 0 & \text{otherwise} \end{cases}, \\ Y_{bls} &:= \begin{cases} 1 & \text{if batch } b \text{ of machine } l \text{ contains only jobs of family } s \\ 0 & \text{otherwise} \end{cases}. \end{aligned}$$

The present scheduling problem is formulated as follows:

$$\min \sum_{j=1}^n w_j \cdot T_j \quad (2)$$

subject to

$$\sum_{b=1}^{n_b} \sum_{l=1}^m X_{jbl} = 1, \quad \forall j = 1, \dots, n, \quad (3)$$

$$\sum_{j=1}^n X_{jbl} \leq B, \quad \forall b = 1, \dots, n_b, l = 1, \dots, m, \quad (4)$$

$$\sum_{s=1}^f Y_{bbs} = 1, \quad \forall b = 1, \dots, n_b, l = 1, \dots, m, \quad (5)$$

$$e_{js} \cdot X_{jbl} \leq Y_{bbs}, \quad \forall j = 1, \dots, n, b = 1, \dots, n_b, l = 1, \dots, m, \quad (6)$$

$$p_s \cdot Y_{bbs} \leq C_{ll}, \quad \forall l = 1, \dots, m, s = 1, \dots, f, \quad (7)$$

$$C_{(b-1)l} + \sum_{s=1}^f p_s \cdot Y_{bbs} \leq C_{bl}, \quad \forall b = 2, \dots, n_b, l = 1, \dots, m, \quad (8)$$

$$(C_{bl} - d_j) - M \cdot (1 - X_{jbl}) \leq T_j, \quad \forall j = 1, \dots, n, b = 1, \dots, n_b, l = 1, \dots, m, \quad (9)$$

$$C_{bl}, T_j \geq 0, X_{jbl}, Y_{bbs} \in \{0, 1\}. \quad (10)$$

The objective (2) intends to minimize TWT. Constraints (3) ensure that each job is assigned to a batch, and constraints (4) do not allow more than B jobs assigned to the same batch. With constraints (5) we make sure that each batch belongs to a single job family, and constraints (6) ensure that the families of the jobs assigned to a batch match the family of the batch. Using constraints (7) the completion times of the first batches on each machine are computed, whereas constraints (8) ensure the correct completion times for all subsequent batches. Finally, constraints (9) express the tardiness for each job. A similar formulation for minimizing the makespan can be found in [6].

Since the problem (1) is NP-hard, calculating the optimal solution of the above model for instances with more than 30 jobs is out of range for standard MIP solvers within a reasonable time.

2.2 Literature Review

Batching problems are considered by many researchers [4]. A survey of batching models in semiconductor manufacturing is provided in [8].

ACO is a population based metaheuristic designed to solve combinatorial optimization problems. ACO type approaches have been applied to different scheduling problems. Bauer *et al.* [2] developed first an ACO algorithm for the single machine total tardiness problem. ACO schemes are applied to problem $1 || TWT$ in [3,9]. ACO type approaches were also suggested for scheduling problems related to parallel machines. Here, usually the assignment and the sequencing decisions are represented by pheromone trails. For example, the problem $P_m || TWT$ is studied using ACO in [13].

So far only some preliminary work related to batch scheduling with artificial ants is described in the literature. An ACO type approach to problem (1) is described in [12]. They use the Apparent Tardiness Cost (ATC) dispatching rule to form batches. ACO is used to assign batches to machines. However, we use a quite different representation that has the advantage to include batch formation, assignment, and sequencing. Li *et al.* [7] considered the scheduling problem $P_m | s_{kj}, r_j, incompatible, p - batch | TWT$ where r_j denotes the ready time of job j . They form batches using the time window technique suggested in [10] for batching problems. An AS type approach is used to assign the batches to machines and sequence them. However, only a few computational results are available. A quite different AS for the single machine scheduling problem $1 | p - batch, incompatible | \sum w_j C_j$ is suggested in [5].

3 Heuristic Approaches

3.1 ATC-BATC Heuristic

A simple list based scheduling approach is described for the sake of completeness. Every time a machine becomes free, one batch from each family is formed, and one of all the formed batches is selected to be scheduled next. The number of batches will be equal to the number of families that contain unscheduled jobs. The batch is chosen based on the ATC dispatching rule. This heuristic is simple but has been known to give reasonable solutions quickly [1]. The ATC index I_{js} for a job j belonging to family s calculated at time t is given below:

$$I_{js}(t) := w_{js} / p_s \exp(-\max(d_{js} - p_s - t, 0) / \kappa \bar{p}). \quad (11)$$

Here, we denote by t the time for decision-making, $\bar{p}$ is the average processing time of the remaining jobs, and κ is a look-ahead parameter. The jobs in each family are ordered in decreasing order of their ATC indices, and a batch of the first B jobs is formed per family. In order to schedule one of these batches the Batched Apparent Tardiness Cost (BATC) rule is used. Therefore, at time t , for all the batches the following index is calculated by $I_{BATC}(x, s, t) := \sum_{j \in B_{xs}} I_{js}(t)$, where $I_{BATC}(x, s, t)$ is the BATC index for batch x of family s at

time t . The batch with the highest BATC index is scheduled. We call the heuristic that forms batches using ATC and sequence the batches by BATC in the remainder of this paper ATC-BATC-H. It is well known that ATC type rules lead to small TWT values when the look-ahead parameters are set appropriately [1]. We consider all $\kappa \in \{0.5, 1.0, \dots, 5\}$, fix one of these values, solve (1) and take finally the value of κ that leads to the smallest TWT value for the given instance and ATC-BATC-H.

Furthermore, we will use a decomposition heuristic to improve the schedules obtained from ATC-BATC-H. From an initial sequence obtained by ATC-BATC-H a sub-sequence of size λ is solved into an optimal sequence (with respect to TWT) by using total enumeration. The first α batches of this optimal sequence are fixed into the final sequence, and the next λ batches are considered. This process is repeated until all batches are covered and no improvement is made on TWT. At maximum *iter* iterations are allowed. We call the heuristic based on the decomposition heuristic DH($\lambda, \alpha, iter$).

We use the swap procedure suggested in [1] for batching with parallel machines in order to improve the quality of the schedules. The swap procedure exchanges jobs for all batches of one family across the machines with the objective to reduce TWT. We denote the swap procedure in the remainder of this paper by Swap.

Note that in [1] batches are formed by using ATC-BATC-H. These batches are assigned to the machines via a GA. DH and Swap is used to improve the schedules. The schedules obtained by using these algorithms lead to the smallest TWT among all the tested heuristics. Therefore, we will use this heuristic for comparison purposes within this research. This approach is called ATC-GA1a-DH+swap in [1]. In this paper, we call it only GA for abbreviation. ATC-BATC-H is used as a reference heuristic within this paper.

3.2 ACS Scheme

We start with describing the used ACS approach. The scheduling problem (1) is represented as a pure sequencing problem denoted as

$$\pi := (\pi[1], \dots, \pi[n]). \quad (12)$$

We construct a permutation by inserting the jobs of a newly formed and chosen batch to the partial sequence starting from an empty sequence and describe how we form appropriate batches.

Therefore, we consider a permutation of the jobs for each family s that are not included in the partial sequence of representation (12) $\pi_s^t := (\pi_s[1], \dots, \pi_s[n_{s,curr}])$, where $\pi_s[i]$ denotes the job of family s that is sequenced on position i in π_s^t and the notation $n_{s,curr}$ is used for the number of jobs that are not already part of the partial schedule that corresponds to (12). The time t denotes the time where the newly formed batch can start with processing.

Starting from π_s^t , we obtain a sequence of batches by considering subsequences of the form $\chi_s(w) := (\pi_s[wB+1], \pi_s[wB+2], \dots, \pi_s[(w+1)B])$, where $w \in \{0, \dots, \lceil n_{s,curr}/B \rceil\}$. Note that only the last batch formed in this way may have less than B jobs. We assume that the batches are sequenced in decreasing order of importance, i.e., $\chi_s(w)$ is always more important than batch $\chi_s(w+1)$. The used measure of importance is related to the ACS approach and is described later. We select the most important batch among the $\chi_s(0), s = 1, \dots, f$ and add it to the partial sequence to construct the job permutation (12).

We use a list based approach to assign the selected batch to the machines. Here, we simple use the machine that is available next to assign the next batch to it. This availability time is the t used in the notation π_s^t . By doing so, the job sequence (12) can be constructed to determine a solution of (1). Each solution of (1) can be transferred in a job sequence (12).

Based on this procedure, it is enough to describe a heuristic to find job permutations that lead to small values of TWT. We describe an ACO type approach to tackle this problem. The main idea of ACO consists in maintaining a set of artificial ants. Each ant starts with an empty sequence of the jobs and constructs a permutation by adding iteratively one of the remaining jobs to the partial sequence constructed so far.

The search for good sequences with respect to scheduling problem (1) is coupled between the different ants by artificial pheromone trails. The ants communicate indirectly by modifying the pheromone trails after the construction of a new permutation π . The solutions for (1) that correspond to the permutation can be further improved by local search techniques.

The construction of a solution corresponds to create a new ant in the general ACO scheme [3]. This ant takes the pheromone trail information into account while its construct a solution of problem (1). It updates the pheromone trail locally whenever it adds a new job to the job permutation (12). When the maximum number of ants per iteration is not reached we construct a new solution. Otherwise, we update pheromone trails globally by using the best solution obtained by the ants so far.

We have to describe the tailoring of the steps of the general ACO scheme to solve (1). We denote by η_{ij} the heuristic desirability for setting $\pi[i] := j$. The η_{ij} values are typically derived by using appropriate problem specific dispatching rules. In this research, we use the ATC dispatching rule with priority index (11). In this situation, t is the time when the next machine gets available.

We denote by $\tau_{ij}(l_{iter})$ the pheromone intensity that is associated with the setting $\pi[i] := j$, i.e., job j is placed on position i in π . The parameter l_{iter} is used for the current iteration of the ACO scheme. The maximum number of ants used within one iteration is denoted by n_{amax} . The construction of a solution within one iteration of the ACO scheme works as follows:

1. Initialize the number of the current ant by $n_a = 1$.
2. Create a new artificial ant. Denote the set of jobs that are not used to construct π by J_U . Initialize the set with $J_U := \{1, \dots, n\}$. We introduce the set of unscheduled jobs $J_U^s := \{j \in J_U \mid j \text{ is part of family } s\}$ for each family s .

3. Create a sample of a $U[0,1]$ distributed random variable. Denote the obtained value by q_0 . We assume that we have at least $n_{s,curr} > 0$ jobs in J_U^s . When $q_0 \leq q$ then the $B_{s,curr} := \min(B, n_{s,curr})$ jobs that maximize the value of

$$\Phi_{ij}^s := \frac{1}{B_{s,curr}} \sum_{g=0}^{B_{s,curr}-1} \left(\sum_{l=1}^{i+g} \tau_{lj+g}(l_{iter}) \right) \eta_{i+g+j+g}^\beta \quad (13)$$

over all families s are set into the current partial sequence on position $i, \dots, i + B_{s,curr} - 1$. A similar pheromone summation rule is suggested in [9] for the scheduling problem $1 || TWT$. Using the sum of pheromone values of job j up to position i offers the advantage that the selection of jobs j that are chosen by previous ants for positions smaller than i will be enforced. The quantity $\beta > 0$ is a parameter of ACS that is related to the importance of the desirability. Here, $q \in [0,1]$ is a fixed number that is a parameter of the ACS. Usually, q is selected close to 1. Hence, the ant makes the best decision with a probability q as indicated by the heuristic information and the pheromone trails. These decisions make sure an exploitation of the search space. When $q_0 > q$ then job $j \in J_U^s$ is selected according to the following discrete distribution with the probabilities p_{ij} given by

$$p_{ij} := \begin{cases} \frac{\left(\sum_{l=1}^i \tau_{lj}(l_{iter}) \right) \eta_{ij}^\beta}{\sum_{r \in J_U^s} \left(\sum_{l=1}^i \tau_{lr}(l_{iter}) \right) \eta_{ir}^\beta} & \text{when } j \in J_U^s \\ 0 & \text{otherwise.} \end{cases} \quad (14)$$

For each family, we select a job according to this discrete probability distribution. We form the batch including this job and the next $B_{s,curr} - 1$ jobs from a list that is sequenced in

decent order according to the index $\left(\sum_{l=1}^i \tau_{lj}(l_{iter}) \right) \eta_{ij}^\beta$. Note that we choose only the first job

of the batch according to (14) because we are not interested in completely destroying the structure of batches formed by ATC-BATC-H type rules. We evaluate the batches for all families according to expression (13) and consider the batch with the largest value Φ_{ij}^s .

The ant performs a biased exploration with a probability of $1 - q$.

4. A local step-by-step update of the pheromone trail is performed immediately after an ant has added all jobs of a batch to the partial sequence by

$$\tau_{ij}(l_{iter}) := (1 - \theta) \tau_{ij}(l_{iter}) + \theta \tau_0, \quad (15)$$

where $\theta \in (0,1]$ and $\tau_0 > 0$ are parameters of the ACS. The local update is necessary to make the decision to put job j on position i less desirable for consecutive ants. The exploration of different sequences is enforced by performing the step-by-step update.

5. Set $J_U := J_U - B_{s,curr}$. When $J_U \neq \emptyset$ go to Step 3. When $J_U = \emptyset$ then improve the schedule by using DH and Swap. Change the position of the jobs in representation (12). Update the number of ants by $n_a := n_a + 1$. Go to Step 2 when $n_a \leq n_{a,max}$ is valid.

6. When job j is scheduled on position i in the global best solution at iteration l_{iter} then update the global pheromone trail by the following expression:

$$\tau_{ij}(l_{iter} + 1) := (1 - \rho)\tau_{ij}(l_{iter}) + \rho/TWT^*, \quad (16)$$

where $\rho \in (0, 1]$ is a parameter of the ACO approach that is used to model the pheromone evaporation. TWT^* is the TWT value of either the global-best or the iteration-best solution. Note that also mixed strategies are possible.

7. Update the iteration number by $l_{iter} := l_{iter} + 1$. Start again with Step 1 when the termination condition is not fulfilled.

3.3 MMAS Scheme

We perform also some additional experimentation with a MMAS. Here, we set basically $q = 0$ in ACS and avoid any local update of the pheromone trails in Step 4 of the ACS scheme. In MMAS, the pheromone trail is bounded by an upper and a lower pheromone value (cf. [14] for a more detailed description of the MMAS principles). We denote these values by τ_{max} and τ_{min} respectively. We use the following settings for the two bounds according to [14] $\tau_{max} := 1/(1 - \rho)TWT^*$, where TWT^* is the TWT value of the iteration-best solution, and $\tau_{min} := 2B\tau_{max}(1 - \sqrt[n]{p_{best}})/(n - 2B)\sqrt[n]{p_{best}}$ because an ant has to choose on average on $n/(2B)$ different jobs. Here, p_{best} is the probability that an ant constructs the best solution by n consecutive right decisions. When p_{best} is chosen too small than we may observe that $\tau_{min} > \tau_{max}$. In this case we have to set $\tau_{min} = \tau_{max}$. If p_{best} is close to one then τ_{min} is also close to zero. Hence, a careful choice of the parameter p_{best} is crucial for the performance of MMAS.

4. Computational Experiments

4.1 Design of Experiments and Implementation

In this section, we present results of computational experiments. We used test instances generated according to an extension of the test data description given in [1] to the case of more parallel machines. The test cases depend on the batch size, the number of parallel machines, the number of jobs of each job family, the due date range R and the tightness T of the schedules. In our experiments we consider the case of three different job families. Totally, we solve 480 different test instances.

All the tests are performed on an Intel Pentium D with 3.2 GHz CPU and 4 GB of random access memory with SUSE Linux 10 as operating system. The algorithms are implemented in the C++ programming language, for the GA implementation the framework Galib is used.

We also solved a couple of test instances for $f = 2, B = 4, m = 2, n = 16$ and for $f = 3, B = 4, m = 3, n = 24$ using the MIP (2)-(10) optimally. These experiments were performed with ILOG CPLEX 11.1. ACS found the optimum solution for all instances.

4.2 Comparison of ACS and GA

We perform experiments with four different amounts of time for computation in order to carry out a fair comparison of ACS and GA. The allowed time for computation is 5, 10, 30, and 60

seconds respectively. In order to select the various parameter of the ACO scheme, we performed extensive experiments on a subset of the generated test instances. Finally, we come up with $n_{a\max} = 5$, $q = 0.9$, $\tau_0 := 1 / TWT_{best}$ where TWT_{best} denotes the best TWT value obtained by ATC-BATC-H. The global-best solution is used for global pheromone update with $\rho = 0.01$. The evaporation parameter for local update is $\theta = 0.1$. Furthermore, we set $\beta = 1$ within all experiments. DH is used for both ACS and GA. In order to compare the performance of the two solution techniques with GA, we simply apply the same parameter setting as [1], i.e., we set $\lambda = 5, \alpha = 2$ and $iter = 15$. The remaining parameters of the GA (except the stopping criteria) are the same as used in [1].

The results are presented in Table 1. We show the ratio of the TWT value of the metaheuristic obtained as average value from three independent runs and the TWT value of ATC-BATC-H. We present the time for computing immediately after the name of the heuristic.

Table 1: TWT Values Relative to ATC-BATC for Different Allowed Computing Times

Level	ACS5	ACS10	ACS30	ACS60	GA5	GA10	GA30	GA60
$n = 180$	0.9196	0.9171	0.9151	<u>0.9133</u>	0.9407	0.9294	0.9178	0.9171
$n = 240$	0.9213	0.9167	0.9122	<u>0.9101</u>	0.9521	0.9383	0.9195	0.9136
$n = 300$	0.9248	0.9212	0.9147	<u>0.9120</u>	0.9601	0.9475	0.9283	0.9195
$B = 4$	0.9255	0.9226	0.9191	<u>0.9166</u>	0.9615	0.9504	0.9324	0.9244
$B = 8$	0.9224	0.9181	0.9130	<u>0.9110</u>	0.9449	0.9307	0.9155	0.9132
$m = 3$	0.9285	0.9256	0.9222	<u>0.9210</u>	0.9556	0.9465	0.9353	0.9316
$m = 4$	0.9194	0.9156	0.9116	<u>0.9093</u>	0.9473	0.9373	0.9222	0.9177
$m = 5$	0.9180	0.9145	0.9097	<u>0.9073</u>	0.9491	0.9332	0.9149	0.9088
$m = 6$	0.9132	0.9091	0.9040	0.9011	0.9432	0.9280	0.9066	<u>0.9007</u>
$R = 0.5$ $T = 0.3$	0.7885	0.7800	0.7697	<u>0.7636</u>	0.8444	0.8148	0.7814	0.7740
$R = 0.5$ $T = 0.6$	0.9641	0.9620	0.9595	0.9582	0.9712	0.9659	0.9585	<u>0.9559</u>
$R = 2.5$ $T = 0.3$	0.9812	0.9797	0.9780	0.9771	0.9898	0.9848	0.9786	<u>0.9767</u>
$R = 2.5$ $T = 0.6$	0.9696	0.9673	0.9647	<u>0.9640</u>	1.0149	1.0043	0.9849	0.9764
Overall	0.9259	0.9223	0.9180	0.9157	0.9551	0.9425	0.9258	0.9207

Looking at the overall results the GA continuously improves with increasing runtime, but ACS reaches after 30 seconds already a better solution quality than the GA after twice that time. Furthermore, the results show that ACS convergence fast to a good solution and the additional solution quality gained by extending the runtime is small. For ACS the difference between 5 and 60 seconds runtime is less than 1%.

Looking at the detailed results for different number of jobs, respectively different number of families, it seems that the performance of the GA drops significantly if the problem gets more complex (more jobs). ACO show this performance difference only for short runtimes of 5 or 10 seconds, but those differences level out for longer runtimes.

Obviously the due dates (parameters R and T) have greatest impact on the solution quality. For test instances with a small range of the due dates ($R=0.5$) together with a low number of tardy jobs ($T=30\%$) the ATC-BATC-H solution can be improved by more than 20%. If the range of the due dates is large ($R=2.5$) or the number of tardy jobs is high ($T=60\%$) the average improvement of both methods is significantly less, but still the ordering $GA > ACO$ with respect to solution quality is the same.

4.3 Computational Experiments with Different Pheromone Update Schemes and with MMAS

In this second series of experiments, we are interested in identifying the performance of MMAS and the impact of using global-best and iteration-best solution for globally updating the pheromone trails within ACS. In both situations, we do not prescribe a maximum amount of computing time. Instead, we allow for at maximum 1000 iterations. The parameter setting for MMAS is similar to ACS, except $\theta = 0$, $q = 0$, $\tau_0 = 10000.0$, and $\beta = 2$. We select $\beta = 5$ when the TWT value obtained by ATC-BATC-H is larger as 1000. The iteration-best solutions are chosen for pheromone trail updates. We also incorporated a pheromone trail smoothing as suggested in [14]. Note that we also performed for experiments with values ρ close to 1. However, we obtained for a majority of test instances better results by using the smaller value. In our experiments, $p_{best} = 0.005$ is chosen.

In Table 2, we show the TWT values for both MMAS and for ACS relative to the corresponding TWT values for ATC-BATC-H. We denote by ACS-M-GB-IB a mixed strategy that allows the global-best and the iteration-best solution to update the pheromone trails with an evaporation parameter $\rho = 0.01$. The abbreviation ACS-IB is used for ACS with iteration-best solution. The value of ρ is equals to 0.01 in this case. Finally, the scheme described in Section 3.2 is called ACS-GB.

Table 2: TWT Values for MMAS and Different Pheromone Update Schemes

Level	MMAS	ACS-M-GB-IB	ACS-IB	ACS-GB
$n = 180$	0.9134	0.9136	0.9120	0.9128
$n = 240$	0.9088	0.9092	0.9072	0.9082
$n = 300$	0.9100	0.9100	0.9094	0.9095
$B = 4$	0.9145	0.9156	0.9145	0.9147
$B = 8$	0.9109	0.9103	0.9086	0.9096
$m = 3$	0.9214	0.9208	0.9185	0.9196
$m = 4$	0.9087	0.9083	0.9072	0.9075
$m = 5$	0.9045	0.9057	0.9044	0.9052
$m = 6$	0.8999	0.9004	0.8996	0.8999
$R = 0.5, T = 0.3$	0.7543	0.7604	0.7589	0.7600
$R = 0.5, T = 0.6$	0.9571	0.9579	0.9576	0.9581
$R = 2.5, T = 0.3$	0.9815	0.9767	0.9755	0.9759
$R = 2.5, T = 0.6$	0.9658	0.9643	0.9619	0.9624
Overall	0.9147	0.9148	0.9135	0.9141

It turns out that differences between MMAS and ACS are small. But the computing time for MMAS is very often several minutes per instance. Furthermore, using the iteration-best solution for the global pheromone update provides only slightly better results. Among the four different strategies using the global-best and the iteration-best solution is the worst one.

5. Conclusions and Future Research

We suggested two ACO approaches for scheduling jobs with incompatible families on parallel identical batching machines. We compared the solution quality and the time needed for computation of the two ACO approaches with results for a GA suggested in previous research [1]. All the approaches were hybridized by a decomposition scheme and local search approaches.

It turned out that the two ACO approaches provide solutions with similar solution quality. The GA approach is slightly outperformed by ACS with respect to solution quality. Among the

three metaheuristics the ACS approach is the fastest one. A small time for computation is especially interesting when using one of the metaheuristic approaches within decomposition approaches like the shifting bottleneck heuristic for entire wafer fabs because in this case we have to solve a large number of scheduling problems for jobs on parallel machines [11].

Multiple order per job (MOJ) scheduling problems are important in 300 mm wafer fabs. Several customer orders have to be assigned to one Front-Opening-Unit-Pod (FOUP). The FOUPS travel around the wafer fab and the wafers of the FOUP will be processed on different machines. The problem of FOUP formation is similar to our batching problem. That is why we believe that we can use algorithms from this research to tackle MOJ scheduling problems.

Acknowledgements The authors gratefully acknowledge the testing efforts of Markus Erhardt.

References

1. Balasubramanian, H., Mönch, L., Fowler, J. W., Pfund, M., Genetic Algorithm Based Scheduling of Parallel Batch Machines with Incompatible Job Families to Minimize Total Weighted Tardiness, *International Journal of Production Research*, 42(8), 1621-1638, (2004)
2. Bauer, A., Bullnheimer, B., Hartl, R., Strauss, C., Minimizing Total Tardiness on a Single Machine Using Ant Colony Optimization, *Central European Journal of Operations Research*, 8(2), 125-141, (2000)
3. den Besten, M., Stützle, T., Dorigo, M., Ant Colony Optimization for the Total Weighted Tardiness Problem, *Proceedings 6th International Conference Parallel Problem Solving from Nature (PPSN VI)*, 611-620, (2000)
4. Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C. N., Tautenhahn, T., van de Velde, S., Scheduling a Batching Machine, *Journal of Scheduling*, 1, 31-54, (1998)
5. Kashan, A. H., Karimi, B., Scheduling a Single Batch Processing Machine with Arbitrary Job Sizes and Incompatible Job Families: an Ant Colony Framework, *Journal of the Operational Research Society*, 1-12, (2007)
6. Klemmt, A., Horn, S., Weigert, G., Hielscher, T. Simulation-based and Solver-based Optimization Approaches for Batch Processes in Semiconductor Manufacturing, *Proceedings of the 2008 Winter Simulation Conference*, 2041-2049, (2008)
7. Li, L., Qiao, F., Wu, Q. ACO-based Scheduling of Parallel Batch Processing Machines with Incompatible Job Families to Minimize Total Weighted Tardiness, *Proceedings ANTS 2008*, 219-228, (2008)
8. Mathirajan, M., Sivakumar, A. I., A Literature Review, Classification and Simple Meta-analysis on Scheduling of Batch Processors in Semiconductor, *International Journal of Advanced Manufacturing Technology*, 29, 990-1001, (2006)
9. Merkle, D., and Middendorf, M. An Ant Colony Optimization Algorithm with a New Pheromone Evaluation Rule for Total Tardiness Problems, *Proceedings of the EvoWorkshops 2000*, LNCS 1803, 287-296, (2000)
10. Mönch, L., Balasubramanian, H., Fowler, J. W., Pfund, M. E., Heuristic Scheduling of Jobs on Parallel Batch Machines with Incompatible Job Families and Unequal Ready Times, *Computers & Operations Research* 32, 2731-2750, (2005)
11. Mönch, L., Schabacker, R., Pabst, D., Fowler, J. W., Genetic Algorithm-based Subproblem Solution procedures for a Modified Shifting Bottleneck Heuristic for Complex Job Shops, *European Journal of Operational Research*, 177(3), 2100-2118, (2007)
12. Srinivasa Raghavan, N. R., Venkataramana, M., Scheduling Parallel Batch Processors with Incompatible Job Families Using Ant Colony Optimization, *Proceedings of the 2006 International Conference on Automation Science and Engineering*, 507-512, (2006)
13. Srinivasa Raghavan, N. R., Venkataramana, M., Parallel Processor Scheduling for Minimizing Total Weighted Tardiness Using Ant Colony Optimization, *Journal of Advanced Manufacturing Technology*. To appear (2009)
14. Stützle, T., Hoos, H. H., MAX-MIN Ant System, *Future Generation Computer Systems* 16, 889-914, (2000)

A heuristic for minimizing weighted carry-over effects in round robin tournaments

Allison C. B. Guedes^{*} · Celso C. Ribeiro^{**}

Abstract The carry-over effects value is one of the various measures one can consider to assess the quality of a round robin tournament schedule. We introduce and discuss a new, weighted variant of the minimum carry-over effects value problem. The problem is formulated by integer programming and an algorithm based on the hybridization of the Iterated Local Search metaheuristic with a multistart strategy is proposed. Numerical results are presented.

Keywords: Carry-over; round robin; tournaments; sports scheduling; timetabling; heuristics; iterated local search

1 Motivation

Sports optimization has been attracting the attention of an increasing number of researchers and practitioners in multidisciplinary areas such as operations research, scheduling theory, constraint programming, graph theory, combinatorial optimization, and applied mathematics. Kendall et al. [1] surveyed state-of-the-art applications and methods for solving optimization problems in sports scheduling. Rasmussen and Trick [2] reviewed scheduling problems in round robin tournaments, which are of special importance due to their practical relevance and interesting mathematical structure.

There are many relevant aspects to be considered in the determination of the best fixture for a tournament. In some situations, one seeks for a schedule minimizing the total traveled distance, as in the case of the traveling tournament problem [3] and in that of its mirrored variant [4], which is common to many tournaments in South

^{*} Supported by the FAPERJ grant E-26/100.497/2008.

^{**} Supported by CNPq grants 301.694/2007-9 and 485.328/2007-0, as well as by FAPERJ grant E-26/102.805/2008.

Instituto de Computação, Universidade Federal Fluminense
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil
E-mail: aguedes@ic.uff.br

Instituto de Computação, Universidade Federal Fluminense
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil
E-mail: celso@ic.uff.br

America. Other problems attempt to minimize the number of breaks, i.e., the number of pairs of consecutive home games or consecutive away games. Ribeiro and Urrutia [5] tackled the scheduling of the Brazilian national football tournament, in which the objective function consists of maximizing the number of games that may be broadcast by open TV channels, so as to increase the revenues from broadcast rights.

The minimization of the *carry-over effects value* [6] is a fairness criterion leading to an even distribution of the sequence of games along the schedule.

A major issue in the strategy of a team (or an athlete), in particular in long competitions, consists of balancing their efforts over the competition. If a team plays against a weak opponent, it is likely to be in better shape to play in the next round than if it had played against a hard opponent before. Teams that play against strong opponents will very likely be more tired for their next game. Therefore, it is likely that a team (or an athlete) makes much less effort playing against an opponent that played before against a very strong contestant, than it would make against an opponent that faced an easy contestant.

The above situation is particularly true in the case of sports which require a great amount of physical effort (such as wrestling, rugby, and martial arts). In this sort of sports, it is not uncommon that a team (or an athlete) plays several games in a row, making a sequence of very tired (resp. well reposed) opponents very attractive (resp. unattractive). Some fixtures may contain several of such sequences of easier or harder games assigned to one or more teams. This situation does not characterize a fair schedule and is highly undesirable in any tournament. To illustrate this effect, suppose a Karate-Do or Judo competition. There is no weight division in open-weight categories of such sports: a physically weak athlete may fight a strong one. A contestant that has just fought a very strong opponent will possibly be very tired (and even wounded) in his/her next fight. This would deteriorate his/her performance, giving to the next opponent a strong advantage that otherwise he/she would not have.

Suppose that team A plays team C right after playing team B. If team B is much stronger than the other competitors, then team C will possibly take some advantage over team A in their game. This is due to the great effort team A has made in its previous game. This type of situation in which one of the teams may be benefited should be avoided or, at least, minimized.

We say that a team C receives a carry-over effect due to team B if there is a team A that plays C just after its game against B. We consider a single round robin tournament played by n (even) teams, in which every team plays each other exactly once. Figure 1 (a) displays a matrix describing a hypothetical tournament fixture, whose entry in row i and column j informs the team playing against team j in round i . One may count the number of carry-over effects each team gives to every other in the fixture of a round robin tournament and then build the *carry-over effects matrix*. Each entry in row i and column j of this matrix indicates the number of carry-over effects team i gives to team j . Figure 1 (b) presents the underlying carry-over effects matrix associated with the fixture represented in Figure 1 (a).

A fair tournament regarding carry-over effects is one in which the latter are evenly distributed over the cells of the carry-over effects matrix and the total carry-over effects value is minimized. The problem of minimizing the carry-over effects value was originally proposed by Russell [6]. In this work, we extend the latter by the minimization of the *weighted carry-over effects value*. This new, weighted problem is introduced and formulated in the next section. Section 3 reviews previous solution approaches for the unweighted original problem and describes a heuristic based on the hybridization of

	A	B	C	D	E	F	G	H		A	B	C	D	E	F	G	H
1	H	C	B	E	D	G	F	A	A	0	0	3	0	1	2	1	0
2	C	D	A	B	G	H	E	F	B	5	0	0	0	1	0	0	1
3	D	E	F	A	B	C	H	G	C	0	1	0	3	0	3	0	0
4	E	F	H	G	A	B	D	C	D	0	2	0	0	2	0	3	0
5	F	G	E	H	C	A	B	D	E	1	1	0	2	0	2	0	1
6	G	H	D	C	F	E	A	B	F	0	0	0	0	2	0	3	2
7	B	A	G	F	H	D	C	E	G	0	3	1	0	0	0	0	3
									H	1	0	3	2	1	0	0	0

(a)

(b)

Fig. 1 (a) Fixture of a tournament with eight teams and (b) its carry-over effects matrix.

the Iterated Local Search metaheuristic with a multistart strategy for approximately solving the weighted variant. Numerical results are reported and discussed in Section 4. Concluding remarks are drawn in the last section.

2 Weighted formulation

A *single round robin tournament* is one in which each team plays every other exactly once. The games take place along a predefined period of time organized into rounds or slots. A *compact tournament* has a minimum number of rounds. A *compact single round robin tournament* with n (even) teams has $n - 1$ rounds.

For a given compact single round robin schedule with n teams, one says that team i gives a carry-over effect to team j if some other team plays consecutively against teams i and j , i.e., some team plays against team i in round t and against team j in round $t + 1$ for some $t \in \{1, \dots, n - 1\}$, where the rounds are considered cyclically (i.e., round $n - 1$ is followed by the first round). If team i is a very strong or very weak team and several teams play consecutively against teams i and j in this order, then team j may be, respectively, handicapped or favored compared with other teams. In an “ideal” schedule with respect to carry-over effects, no teams i, j, x, y should exist such that teams x and y both play against team j immediately after playing against team i .

The carry-over effects matrix $C = (c_{ij})$, with $i, j = 1, \dots, n$, has been introduced to measure the balance of a tournament with respect to this criterion. Each entry c_{ij} of this matrix counts the number of carry-over effects given by team i to team j . It can be seen that $c_{ii} = 0$ and $\sum_{j=1}^n c_{ij} = \sum_{i=1}^n c_{ij} = n - 1$ for all rows and columns $i, j = 1, \dots, n$. The quality of a schedule with respect to carry-over effects is measured by the carry-over effects value $\sum_{i=1}^n \sum_{j=1}^n c_{ij}^2$. An ideal or balanced schedule is one in which the lower bound value $n(n - 1)$ is achieved, i.e., all non-diagonal elements of the corresponding matrix C are equal to one.

The original (unweighted) carry-over effects value minimization problem does not consider any information with respect to the relative strengths of the teams: all carry-over effects have the same unit weight. However, in real sports competitions it is very likely that the organizers do have an initial estimate of how well a specific team should perform in the competition, considering e.g. its ranking in last year’s competition, its traditions, the quality of its players, and the support of its fans in its home games. Noronha et al. [7] provide a concrete example of a competition (the Chilean football tournament) in which the teams are classified a priori according to some of these criteria in the effort to build a fair schedule.

Therefore, one may conclude that the minimization of the carry-over effects value does not guarantee a fair schedule. In fact, a broader approach consists of assigning a weight w_{ij} to every ordered pair (i, j) of teams, based on their relative strengths or handicaps, and minimizing the total weighted carry-over effects value.

For every pair of teams $i, j = 1, \dots, n$ (with $i \neq j$) and for every round $k = 1, \dots, n$ (with rounds cyclically represented, such as that the round $n - 1$ is followed by the first round), we define the binary variable $y_{kij} = 1$ if and only if team i plays against team j in round k , with $y_{kij} = 0$ otherwise. We also define the number of carry-over effects given by team i to team j as

$$z_{ij} = \sum_{\ell=1}^n \sum_{k=1}^{n-1} y_{k\ell i} \cdot y_{(k+1)\ell j}, \quad (1)$$

for $i \neq j$; $z_{ij} = 0$ if $i = j$. The integer programming formulation of the weighted carry-over effects value minimization problem is presented below:

$$\min \sum_{i=1}^n \sum_{j=1}^n w_{ij} \cdot z_{ij}^2 \quad (2)$$

$$y_{kij} = y_{kji}, \quad i, j, k = 1, \dots, n \quad (3)$$

$$\sum_{i=1}^n y_{kij} = 1, \quad j, k = 1, \dots, n \quad (4)$$

$$\sum_{k=1}^{n-1} y_{kij} = 1, \quad i, j = 1, \dots, n : i \neq j \quad (5)$$

$$y_{kii} = 0, \quad i, k = 1, \dots, n \quad (6)$$

$$y_{nij} = y_{1ij}, \quad i, j = 1, \dots, n \quad (7)$$

$$y_{kij} \in \{0, 1\}, \quad i, j, k = 1, \dots, n. \quad (8)$$

The objective function (2) minimizes the weighted sum of carry-over effects. Constraints (3) ensure that variables $y_{kij} = y_{kji}$ are the same or, alternatively, that the game between teams i and j is the same as that between teams j and i (there is a unique game between teams i and j in a single round robin tournament). Constraints (4) enforce that every team plays exactly once in each round of a compact schedule. Constraints (5) and (6) guarantee that each team plays exactly once against every other team. Constraints (7) enforce that the n -th round is equivalent to the first. Constraints (8) impose the binary requirements on the variables.

3 Hybrid heuristic for weighted minimization

Russell [6] proposed a construction algorithm for the unweighted problem that generates fixtures matching the lower bound to the carry-over effects value when n is a power of two. The method proposed by Anderson [8] obtained solutions that are still the best known to date for unweighted instances. It makes use of algebraic structures called *starters* [9] to generate schedules. However, the approach presumes that a suitable starter is known beforehand, which may imply in large computation times.

Trick [10] developed a constraint programming method that made it possible to prove the optimality of Russell's method for $n = 6$. Henz, Müller, and Thiel [11] improved the solution obtained by the previous approach for $n = 12$, also using constraint programming. Miyashiro and Matsui [12] developed a time-consuming heuristic based on random permutations of the rounds of fixtures created by the polygon method [13]. They reported more than two days of computation time for $n \geq 18$.

The above algorithms do not explore different strategies to generate the initial fixtures, which limits the quality of the solutions they find. Furthermore, some of them are computationally expensive and not appropriate for the weighted variant.

We propose a tailored heuristic for the minimization of weighted carry-over effects. This heuristic is based on the hybridization of the Iterated Local Search (ILS) meta-heuristic [14, 15] with a multistart strategy. It has two main steps: a multistart phase and an ILS phase. A complete run comprises a number of independent sequences of these two steps and returns the best solution found. The multistart phase generates 100 initial solutions, each of them obtained by a constructive method followed by a local search procedure. The best solution found during the multistart phase is used as the starting point for the ILS phase. The main steps of the pseudo-code of the hybrid heuristic corresponding to Algorithm 1 are described in detail in the next sections.

Algorithm 1 Hybrid heuristic

```

1: for iteration = 1 to 10 do
2:   Build a starting fixture;
3:   repeat
4:     Generate a new initial solution by applying a constructive method;
5:     Improve the initial solution by local search;
6:     Update the best initial solution;
7:   until 100 initial solutions are generated;
8:   Set  $S$  as the best initial solution;
9:   repeat
10:    Obtain a new solution  $S'$  by applying a perturbation to  $S$ ;
11:    Apply local search to solution  $S'$ ;
12:    Replace the current solution  $S$  by  $S'$  using an acceptance criterion;
13:    Update the best known solution  $S^*$ ;
14:   until a stopping criterion is reached
15: end for
16: return  $S^*$ ;

```

3.1 Construction method to build initial solutions

A *factor* of a graph $G = (V, E)$ is a subgraph $G' = (V, E')$ of G , with $E' \subseteq E$. A factor G' is a *1-factor* if all its nodes have their degrees equal to one. A *factorization* F of G is a set of edge-disjoint factors of G , such that the union of their edge-sets is equal to E . A factorization of G formed exclusively by 1-factors is said to be a *1-factorization*. In an *ordered 1-factorization* of G , its 1-factors are taken in a fixed order.

There is a one-to-one correspondence between 1-factorizations and round robin schedules. If each team is assigned to a vertex of G , each edge of the latter corresponds to a game. The games in each round correspond to the edges of a 1-factor, making the entire ordered 1-factorization equivalent to the complete tournament fixture.

The rounds of a tournament schedule can be freely permuted without violating any of its properties. Therefore, new schedules can be generated by picking the rounds of a given schedule in any possible order.

The construction method first generates one starting fixture in line 2 of Algorithm 1, using any of the two approaches described below. The first approach is the well known *polygon method* [13], which gives the so-called canonical 1-factorization [16]. It can be used for any value of n . Assuming the teams are numbered as $1, \dots, n$, the edge-set of the 1-factor corresponding to round k is given by $\{(k, n)\} \cup \{(a(k, \ell), b(k, \ell)) : \ell = 1, \dots, n/2 - 1\}$, for $k = 1, \dots, n - 1$, with

$$a(k, \ell) = \begin{cases} k + \ell, & \text{if } (k + \ell) < n, \\ k + \ell - n + 1, & \text{if } (k + \ell) \geq n, \end{cases} \quad (9)$$

and

$$b(k, \ell) = \begin{cases} k - \ell, & \text{if } (k - \ell) > 0, \\ k - \ell + n - 1, & \text{if } (k - \ell) \leq 0. \end{cases} \quad (10)$$

The second approach [16, 17] can be applied whenever n is a multiple of four. It first separates the teams in two sets $V_1 = \{1, \dots, n/2\}$ and $V_2 = \{n/2 + 1, \dots, n\}$. The first $n/2$ rounds are made up only by games with one team in V_1 and the other in V_2 . The games scheduled for round k correspond to the edges of the factor whose edge-set is defined as $\{(\ell, c(k, \ell)) : \ell = 1, \dots, n/2\}$, for $k = 1, \dots, n/2$, with

$$c(k, \ell) = (k + \ell - 2) \bmod (n/2) + n/2 + 1. \quad (11)$$

Each of the last $n/2 - 1$ remaining rounds is formed by picking and putting together one 1-factor from a 1-factorization of the complete graph associated with V_1 and one 1-factor from a 1-factorization of the complete graph associated with V_2 .

The loop in lines 3 to 7 builds each of the 100 initial solutions in line 4 as follows. First, two rounds of the starting fixture are randomly selected and used as the two first rounds of the new solution. In each subsequent step, one still unused round of the starting fixture is selected to be used in the current, incomplete fixture. This process continues, until a complete solution is obtained. The rules used for selecting and placing rounds of the starting fixture in the incomplete solution under construction mimic those in the well known *nearest neighbor* and *arbitrary insertion* heuristics for the traveling salesman problem [18, 19]:

- *nearest neighbor heuristic*: select the unused round which causes the minimum increment in the weighted carry-over effects value and place it as the last in the solution under construction.

- *arbitrary insertion heuristic*: randomly select any unused round and insert it between the two rounds which minimize the increase in the weighted carry-over effects value.

One of these strategies is randomly chosen with equal probability to generate a new initial solution at each iteration of the multistart phase.

3.2 Local search

The initial solutions built by the construction method described in the previous section are tentatively improved by local search in line 5 of Algorithm 1. We use a local search procedure following the Variable Neighborhood Descent (VND) [20, 21] strategy. The best improving move in each neighborhood is applied to the current solution. The following neighborhood structures described by Costa et al. [22] (see also [4]) are used in this order:

- *Team swap* (TS): a move in this neighborhood corresponds to swapping all opponents of a given pair of teams over all rounds, which is the same as swapping two columns of the matrix representing the fixture.
- *Round swap* (RS): a move in this neighborhood consists of swapping all games of a given pair of rounds, which is the same as swapping two rows of the matrix representing the fixture.
- *Partial team swap* (PTS): for any round r and for any two teams t_1 and t_2 , let S be a minimum cardinality subset of rounds including round r in which the opponents of teams t_1 and t_2 are the same. A move in this neighborhood corresponds to swapping the opponents of teams t_1 and t_2 over all rounds in S .
- *Partial round swap* (PRS): for any team t and for any two rounds r_1 and r_2 , let U be a minimum cardinality subset of teams including team t in which the opponents of the teams in U in rounds r_1 and r_2 are the same. A move in this neighborhood consists of swapping the opponents of all teams in U in rounds r_1 and r_2 .

If the locally optimal solution (with respect to neighborhoods TS, RS, PTS, and PRS) obtained by the VND local search strategy improves the best initial solution, then the latter is updated in line 6.

3.3 Iterated local search

The best initial solution obtained at the end of the multistart phase is copied in line 8 of Algorithm 1 to be used as the starting solution S by the ILS phase in lines 9-14, until some stopping criterion is met.

Each ILS iteration starts in line 10 by a perturbation of the current solution S . The perturbation consists of a sequence of random moves within the *game rotation* (GR) neighborhood introduced by Ribeiro and Urrutia [4]. For each move, a randomly generated game is enforced to be changed from the round where it is currently played to a different, randomly selected round. This first change is followed by a sequence of modifications leading to a feasible solution S' , according to the ejection chain mechanism proposed by Glover [23].

A bias is introduced in the perturbation procedure, to drive the move away from pure randomness. Although one cannot know beforehand the effect of a complete ejection chain move on the weighted carry-over effects value of a solution, the increment (or decrement) implied by the first step of the chain can be easily evaluated. The bias consists simply of enforcing the game with the smallest first increment.

The same local search procedure described in Section 3.2 is applied to the perturbed solution S' in line 11. The current solution S is replaced by S' in line 12 if the carry-over effects value of the latter is less than or equal to $(1+b)$ times the carry-over effects value of the former. The value of parameter b doubles after every $2n$ iterations without the modification of the current solution. It is reset to its initial value whenever the current solution is updated, following the same strategy already used in [4]. Finally, the best known solution S^* is updated in line 13. The ILS phase stops if line 14 detects that a maximum number of deteriorating moves have been accepted since the last time the best solution was updated.

4 Computational results

Each instance is defined by the number of teams and by a weighted matrix of carry-over effects. Four classes of weighted instances have been generated for the computational experiments:

- random instances: weights are randomly generated in the interval $[1, 2n]$. Three matrices identified by letters A, B and C have been generated for each value of n .
- linear instances: a strength in the interval $[1, n]$ is assigned to each team. For simplicity, the strength of team i is made equal to i . Each weight w_{ij} is defined as the absolute value of the difference between the strengths of teams i and j .
- perturbed linear instances: each weight of a linear instance is increased by an individual perturbation, randomly generated in the interval $[-n/2, n/2]$. Three different instances identified by letters A, B and C are generated for each value of n . Absolute values are taken whenever the perturbation leads to a negative weight.
- real-life inspired instances: these six instances are derived from the last six issues of the Brazilian football championship. The strength of each team is given by the number of points it obtained in the previous year. As for the linear instances, each weight w_{ij} is defined as the absolute value of the difference between the strengths of teams i and j .

Data of the weighted instances is available from the authors for benchmarking purposes at http://www.ic.uff.br/~celso/grupo/Weighted_carry-over_instances.zip. We also considered instances made up only of unit costs, which are equivalent to those of the original, unweighted problem.

The computational experiments were performed on an AMD Athlon 64 X2 machine with 2.3 GHz and one GB of RAM memory. The code was implemented in C++ and compiled with the GNU C/C++ compiler (GCC) version 4.2.4 under Ubuntu Linux 8.04.

4.1 Tuning

The strategy used to generate the starting fixtures sensibly changes the quality of the solutions found. We tested two different alternatives to obtain the starting fixture

whenever n is a multiple of four. In the first, only the polygon method is used and canonical factorizations are produced. In the second, the two methods described in Section 3.1 are indistinctly employed to obtain 1-factorizations.

We observed from preliminary results that the main parameters influencing the behavior of the ILS phase are the maximum number of deteriorating moves accepted before termination of this phase, the number of ejection chain (game rotation) moves applied in each perturbation, and the initial value for b . We also observed that reasonable values for these parameters are:

$$\begin{aligned} &\text{maximum number of deteriorating moves: } \{10, 100, 200\} \\ &\text{number of ejection chain moves in a perturbation: } \{1, 5, 10\} \\ &\text{initial value for } b: \{0.01, 0.05, 0.10\} \end{aligned} \quad (12)$$

To assess the behavior of the algorithm with respect to parameter values and implementation strategies, we ran the algorithm on three unweighted instances for which n is a multiple of four – namely, for $n = 12, 16$, and 20 . For each of these instances, we tested every possible combination of the above parameter values with the two alternatives for generating the initial fixture. For each configuration, we took average results over three independent runs.

We first discuss the best strategy to generate the initial fixture. Table 1 displays the results obtained by each of the two alternatives for each problem size: average solution value, best solution value, average computation time (in seconds), and longest computation time (in seconds) over the three runs for each combination of the parameter values. The upper part of this table gives the results for the first alternative, based exclusively on canonical factorizations. The lower part reports results for the second alternative, which makes use of both approaches. Although the first alternative (upper part) is less time consuming than the second, the latter (lower part) seems to find better (or comparable) solutions. In consequence, we selected the second alternative to generate the initial fixture.

Strategy	n	COEV (avg.)	COEV (best)	Time (avg.)	Time (max.)
Canonical factorizations	12	192.000	192.000	3.519	4.000
	16	306.914	304.444	178.765	183.296
	20	489.852	488.444	20.790	21.296
Both factorizations	12	167.630	164.370	64.790	69.111
	16	267.630	258.519	244.827	263.926
	20	492.000	490.370	54.481	62.481

Table 1 COEV values and computation times in seconds for two different strategies for the generation of initial fixtures.

To choose the most appropriate configuration of parameter values, we ran the hybrid heuristic (using the second alternative to generate the initial fixture) with all of the 27 possible combinations of parameter values. We collected minimum and maximum results obtained over three runs of each instance tested with $n = 12, 16$, and 20 . For each of them, the results found with each combination were normalized to the interval $[0, 1]$. Finally, we computed average normalized results over all instances considered. The plot in Figure 2 presents the average normalized carry-over effects value and the average normalized running time for each combination. Similarly, the plot in Figure 3 displays

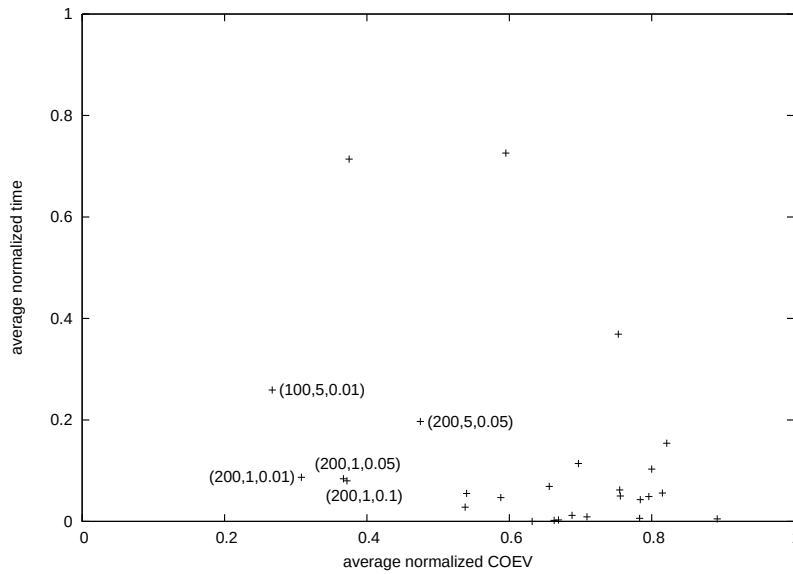


Fig. 2 Average normalized results obtained with different combinations of parameters.

the average normalized best carry-over effects value and the average normalized longest computation time for each combination.

Each point in these plots represents a unique combination of parameter values. The five winning combinations closest to the ideal point $(0,0)$ are the same for both plots. They are identified and labeled with the corresponding maximum accepted number of deteriorating moves, number of ejection chain moves in a perturbation, and initial value of parameter b (in this order).

Since four out of the five best combinations accept a maximum of 200 deteriorating moves before termination of the ILS phase, this value was selected for the first parameter. The number of ejection chain moves in a perturbation is set to one, since three out of the five best points plotted correspond to this value. The best choice for the initial value of parameter b is less evident from these results. For sake of robustness, we selected $b = 0.01$.

4.2 Weighted instances

The hybrid heuristic was run five times for each instance, with the parameter values and implementation strategies as selected above. Average and best carry-over effects values and average and worst computation times in seconds over five runs are reported for each class of weighted test instances.

Results for the random instances are shown in the upper part of Table 2. These instances present the smallest computation times over all classes. The ILS phase was always able to improve the solutions found in the multistart phase, except for the three smallest instances.

We consider next the results for linear instances, reported in the lower part of Table 2. The ILS phase improved the solutions found in the multistart phase for all

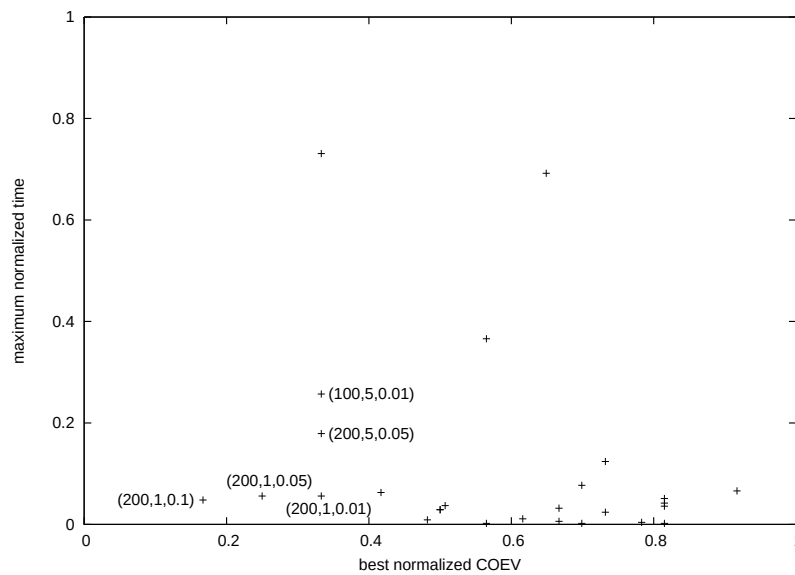


Fig. 3 Extremal normalized results obtained with different combinations of parameters.

instances with $n \geq 8$. The computation times are much greater than those observed for the previous class of instances. This seems to be particularly true for the instance with 20 teams.

Results for perturbed linear instances are shown in the upper part of Table 3. As for the two previous classes, the ILS phase always improved the best solution found in the multistart phase, except for the three smallest instances. The computational times are greater than those observed for the random and linear classes.

Finally, the results obtained for the set of real-life inspired instances are shown in the lower part of Table 3. This class presents the largest computation times for the instances with $n = 20$. This seems to be due to the fact that the weights in this class are more diverse than in any other.

4.3 Validation: unweighted instances

To validate the proposed hybrid heuristic, we also applied it to instances of the original unweighted problem. These instances are equivalent to weighted instances with unit weights. Figure 4 displays the best known solution values known to date and that obtained by the hybrid ILS heuristic for $n = 4, \dots, 16$. The heuristic matched the best known results to date for $n = 4, 6, 8, 10$, and 16. Although it was not able to match the best known solution for $n = 14$, we observe that it improved the best known upper bound to date for $n = 12$ by almost 10%, producing a new, previously unknown solution whose carry-over effects value is 160.

The ILS phase of the heuristic terminates once 200 deteriorating moves have been accepted without improvement in the best solution found. For all classes of weighted instances, we observed that this counter very rarely reached values greater than 50. However, a very different behavior was noticed for the original, unweighted instances.

Instance	Avg. COEV	Best COEV	Avg. time	Worst time
inst4randomA	63.0	63.0	0.2	1.0
inst4randomB	53.0	53.0	0.2	1.0
inst4randomC	45.0	45.0	0.0	0.0
inst6randomA	233.0	233.0	0.8	1.0
inst6randomB	274.0	274.0	0.8	1.0
inst6randomC	235.0	235.0	1.2	2.0
inst8randomA	505.0	505.0	7.2	8.0
inst8randomB	495.0	495.0	18.6	20.0
inst8randomC	470.0	470.0	13.8	14.0
inst10randomA	912.6	895.0	139.4	150.0
inst10randomB	793.0	781.0	151.0	173.0
inst10randomC	744.6	737.0	149.0	172.0
inst12randomA	1549.2	1521.0	453.2	560.0
inst12randomB	1520.2	1489.0	402.4	451.0
inst12randomC	1594.4	1572.0	455.8	546.0
inst14randomA	2620.0	2608.0	39.8	44.0
inst14randomB	2903.8	2870.0	38.0	45.0
inst14randomC	2777.6	2760.0	36.8	42.0
inst16randomA	3812.2	3756.0	2605.8	2878.0
inst16randomB	3846.6	3797.0	2848.2	3294.0
inst16randomC	3773.6	3755.0	3138.0	3483.0
inst18randomA	5574.4	5515.0	1381.2	2442.0
inst18randomB	5816.2	5745.0	792.4	1271.0
inst18randomC	5598.0	5548.0	1547.2	3157.0
inst20randomA	7764.4	7760.0	136.8	145.0
inst20randomB	7928.0	7888.0	147.8	174.0
inst20randomC	7700.2	7636.0	136.8	144.0
Averages	2577.8	2555.8	542.3	711.8
inst4linear	20.0	20.0	0.2	1.0
inst6linear	114.0	114.0	0.6	1.0
inst8linear	168.0	168.0	8.4	9.0
inst10linear	318.0	318.0	64.6	70.0
inst12linear	504.0	496.0	271.4	309.0
inst14linear	960.0	958.0	18.6	20.0
inst16linear	1088.0	1076.0	2007.2	2313.0
inst18linear	1698.0	1660.0	4102.0	4729.0
inst20linear	2257.6	2212.0	5213.0	5624.0
Averages	792.0	780.2	1298.4	1452.9

Table 2 Results for random (upper part) and linear (lower part) instances.

In this case, many improvements obtained during the ILS phase were achieved after more than 50 deteriorating moves.

5 Concluding remarks

We discussed possible applications of the minimum carry-over effects value minimization problem as a fairness criterion to build good fixtures for round robin tournaments. A new, weighted version of the problem was introduced and formulated by integer programming, in which a weight is assigned to each pair of teams.

A hybrid heuristic based on the combination of the Iterated Local Search meta-heuristic with a multistart strategy was proposed and applied to four classes of weighted problem instances with up to 20 teams. The weighted test instances are available from the authors for benchmarking purposes. Numerical results obtained for the original,

Instance	Avg. COEV	Best COEV	Avg. time	Longest time
inst4perturbedlinearA	16.0	16.0	0.2	1.0
inst4perturbedlinearB	17.0	17.0	0.2	1.0
inst4perturbedlinearC	22.0	22.0	0.0	0.0
inst6perturbedlinearA	68.0	68.0	1.4	2.0
inst6perturbedlinearB	73.0	73.0	0.6	1.0
inst6perturbedlinearC	60.0	60.0	0.8	1.0
inst8perturbedlinearA	137.0	137.0	31.0	32.0
inst8perturbedlinearB	141.0	141.0	22.4	23.0
inst8perturbedlinearC	162.0	162.0	26.8	28.0
inst10perturbedlinearA	329.0	326.0	136.0	162.0
inst10perturbedlinearB	277.0	274.0	134.0	163.0
inst10perturbedlinearC	291.8	284.0	128.6	152.0
inst12perturbedlinearA	601.2	587.0	484.2	638.0
inst12perturbedlinearB	528.6	525.0	494.6	580.0
inst12perturbedlinearC	486.6	478.0	546.6	648.0
inst14perturbedlinearA	940.0	920.0	42.8	45.0
inst14perturbedlinearB	947.4	932.0	39.8	43.0
inst14perturbedlinearC	993.2	990.0	36.2	40.0
inst16perturbedlinearA	1403.0	1376.0	3432.8	3905.0
inst16perturbedlinearB	1360.2	1348.0	3343.8	3668.0
inst16perturbedlinearC	1118.0	1098.0	3961.0	4683.0
inst18perturbedlinearA	2063.4	2005.0	5162.4	6983.0
inst18perturbedlinearB	1965.4	1921.0	4038.0	5343.0
inst18perturbedlinearC	1712.0	1585.0	4833.2	5805.0
inst20perturbedlinearA	3092.2	3065.0	482.0	1304.0
inst20perturbedlinearB	2866.8	2800.0	2923.4	4719.0
inst20perturbedlinearC	2837.2	2791.0	1479.6	3509.0
Averages	907.74	888.9	1177.1	1573.3
inst24brasileirao2003	7730.4	7542.0	13897.8	15755.0
inst24brasileirao2004	7179.6	7088.0	12992.8	13468.0
inst22brasileirao2005	5228.8	5158.0	10599.0	13959.0
inst20brasileirao2006	5310.0	5236.0	5705.4	6358.0
inst20brasileirao2007	4876.0	4834.0	2715.8	3655.0
inst20brasileirao2008	4045.6	3944.0	6805.6	8308.0
Averages	5728.4	5633.7	8786.1	10250.5

Table 3 Results for perturbed linear (upper part) and real-life inspired (lower part) instances.

n	Best to date	Hybrid
4	12	12
6	60	60
8	56	56
10	108	108
12	176	160 (new best)
14	234	254
16	240	240

Table 4 Results for the unweighted instances.

unweighted instances contributed to validate the effectiveness of the hybrid heuristic, which was even able to improve the best known solution to date for $n = 12$.

These results confirm previous successful cases of the hybridization of the Iterated Local Search metaheuristic with multistart strategies (e.g., as reported in [4, 24, 25]), in particular in the context of scheduling problems in sports.

References

1. Kendall, G., Knust, S., Ribeiro, C.C., Urrutia, S.: Scheduling in sports: An annotated bibliography (2008)
2. Rasmussen, R.V., Trick, M.A.: Round robin scheduling – A survey. *European Journal of Operational Research* **188**, 617–636 (2008)
3. Easton, K., Nemhauser, G., Trick, M.A.: The travelling tournament problem: Description and benchmarks. In: T. Walsh (ed.) *Principles and Practice of Constraint Programming, Lecture Notes in Computer Science*, vol. 2239, pp. 580–585. Springer (2001)
4. Ribeiro, C.C., Urrutia, S.: Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research* **179**, 775–787 (2007)
5. Ribeiro, C.C., Urrutia, S.: Scheduling the Brazilian soccer tournament with fairness and broadcast objectives. In: *Practice and Theory of Automated Timetabling VI, Lecture Notes in Computer Science*, vol. 3867, pp. 147–157. Springer, Berlin (2007)
6. Russell, K.G.: Balancing carry-over effects in round robin tournaments. *Biometrika* **67**, 127–131 (1980)
7. Noronha, T.F., Ribeiro, C.C., Duran, G., Souyris, S., Weintraub, A.: A branch-and-cut algorithm for scheduling the highly-constrained Chilean soccer tournament. In: *Practice and Theory of Automated Timetabling VI, Lecture Notes in Computer Science*, vol. 3867, pp. 174–186. Springer (2007)
8. Anderson, I.: *Combinatorial Designs and Their Applications*, chap. Balancing carryover effects in tournaments, pp. 1–16. CRC Research Notes in Mathematics. Chapman & Hall (1999)
9. Dinitz, J.H.: *The CRC Handbook of Combinatorial Designs*, chap. Starters, pp. 467–473. The CRC Press Series on Discrete Mathematics and its applications. CRC Press, Boca Raton (1996)
10. Trick, M.A.: A schedule-then-break approach to sports timetabling. In: E. Burke, W. Erben (eds.) *Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III, Lecture Notes in Computer Sciences*, vol. 2079, pp. 242–253. Springer-Verlag (2000)
11. Henz, M., Mller, T., Thiel, S.: Global constraints for round robin tournament scheduling. *European Journal of Operational Research* **153**, 92–101 (2004)
12. Miyashiro, R., Matsui, T.: Minimizing the carry-over effects value in a round robin tournament. In: *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, pp. 460–463. Brno (2006)
13. Kirkman, T.: On a problem in combinations. *Cambridge Dublin Math Journal* **2**, 191–204 (1847)
14. Martin, O.C., Otto, S.W., Felten, E.W.: Large-step markov chains for the traveling salesman problem. *Complex Systems* **5**, 299–326 (1991)
15. Lourenço, H.R., Martin, O.C., Stutzle, T.: *Handbook of Metaheuristics*, chap. Iterated Local Search, pp. 321–353. Kluwer Academic Publishers (2003)
16. de Werra, D.: Scheduling in sports. In: P. Hansen (ed.) *Studies on Graphs and Discrete Programming, Annals of Discrete Mathematics*, vol. 11, pp. 381–395. North Holland (1981)
17. de Werra, D.: Geography, games and graphs. *Discrete Applied Mathematics* **2**, 327–337 (1980)
18. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.): *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons (1985)
19. Gutin, G., Punnen, P. (eds.): *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers (2002)
20. Mladenovic, N., Hansen, P.: Variable neighborhood search. *Computers and Operations Research* **34**, 1097–1100 (1997)
21. Hansen, P., Mladenovic, N.: *Handbook of Metaheuristics*, chap. Variable Neighborhood Search, pp. 145–184. Kluwer Academic Publishers (2002)
22. Costa, F.N., Urrutia, S., Ribeiro, C.C.: An ILS heuristic for the traveling tournament problem with fixed venues. In: E.K. Burke, M. Gendreau (eds.) *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling* (2008)
23. Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics* **65**, 223–253 (1996)
24. Duarte, A.R., Ribeiro, C.C., Urrutia, S., Haesler, E.: Referee assignment in sports leagues. In: *Practice and Theory of Automated Timetabling VI, Lecture Notes in Computer Science*, vol. 3867, pp. 158–173. Springer (2007)

25. Lucena, A.P., Ribeiro, C.C., Santos, A.C.: A hybrid heuristic for the diameter constrained minimum spanning tree problem. *Journal of Global Optimization* (to appear)

Empowerment-based Workforce Scheduling Problem

Abdullah Alsheddy · Edward Tsang

Abstract The Workforce Scheduling Problem (WSP) concerns the construction of optimized work schedules for staff so as to meet customer demands in a cost effective manner while satisfying a wide range of constraints. An effective solution to the problem is critical to companies' performance and success. Organizations are becoming increasingly aware of the importance of enhancing employee empowerment and involvement in decision making. However, the incorporation of empowerment concept in traditional models of the WSP has been scarcely investigated, despite the fact that employee efficiency is crucial to the effectiveness of any workforce scheduling system. The great benefits promised by empowerment management concept motivated us to investigate the deployment of empowerment in designing the workforce scheduling system. This paper proposes a new model that applies a constraint satisfaction approach in order to incorporate the empowerment management strategy. The new model provides employees with an explicit empowerment practice through which they can plan their own specific schedule. A prototype has been developed and its efficiency has been evaluated.

1 Introduction

The workforce scheduling system is a key supportive decision-making system used by organizations to effectively manage human resources [1]. The underlying scheduling process involves several optimization problems, including shift, days-off and tour scheduling, and task assignment.

The Workforce Scheduling Problem (WSP) that we consider here concerns the allocation of jobs to a workforce in an efficient manner, whilst satisfying a large number of operational constraints. It involves both job assignments and routing. It generalizes the well-known Resource Allocation Problem and Vehicle Routing Problem. Several real-life requirements are included, such as multi-skilled staff with various working hours, tasks with different priorities and time-window constraint, route restrictions, and multiple depots.

The School of Computer Science and Electronic Engineering
University of Essex
E-mail: aalshe,edward@essex.ac.uk

This problem occurs in a wide range of service companies such as telecommunications, utilities and other networks, in which the field service engineer is a key resource to be managed.

Employee efficiency is highly critical to the effectiveness of the schedules produced by workforce scheduling systems. This is because time is a coordinate in the process. Unless employees are highly motivated and efficient, they could easily introduce delays which would significantly impact upon the schedule of subsequent tasks, as well as the overall schedule.

Nevertheless, conventional workforce scheduling systems tend to apply traditional management techniques which are based on the command-and-control management strategy [2]. These systems, which we refer to as command-based workforce scheduling systems, isolate employees from the decision making, and leave them powerless in this critical system.

Employee empowerment is another management concept, through which employees are given more control over decisions related to their work. It has been argued that empowerment has promising benefits for both the organization and its employees, offering a win-win approach [3] [4] [2]. Thus, more organizations understand the benefits of enhancing, rather than minimizing, employees' contribution, power and control, with the desire to increase productivity and quality, as well as enhance employee motivation and retention.

Involving employees in the allocation decision within the workforce scheduling systems could incorporate the great benefits promised by empowerment, as suggested by Tsang et al [5]. It could also provide another motivation source for employees, which are normally limited to pay and bonus schemes.

We propose a new model for designing workforce scheduling systems. The new system, which we refer to as empowerment-based workforce scheduling systems, attempts to implement the empowerment management concept by deploying a Constraint Satisfaction [6] approach to provide employees with a form of involvement in the decision making, by which they can suggest their own specific schedules.

Whilst we have chosen to examine the model in relation to a particular WSP, it is believed that this model has the potential to be generalised for use with other staff-job allocation problems. Moreover, it could be extended to other variants of personnel resource allocation problems such as timetabling.

This article is organized into seven sections. Command-based (i.e. traditional) WSP will be described and formulated in section 2. Section 3 will introduce empowerment as a management concept. We will discuss and formulate the Empowerment-based WSP in section 4. The prototype will be explained and the experimental results will be discussed in sections 5 and 6. The conclusion and suggestions for future work will be in section 7.

2 Command-based WSP

Conventional workforce scheduling systems apply a command-and-control strategy in managing human resources, where the system assigns jobs to an employee who has only to accept the decision. The underlying optimization problem focuses mainly on optimizing the organization's objective(s). This focus excludes employees from the decision making, as their involvement could affect the optimality of the organization's objective(s).

Problem formulation is a very important step which is often underestimated [7,8]. Modelling formalizes the problem's considerations and objectives. Below is an attempt to rigorously formulate the Command-based WSP.

2.1 Problem Formulation

The traditional Workforce Scheduling Problem (*command-base WSP*) is basically the problem of allocating a set of technicians (resources): $R = \{r_1, r_2, \dots, r_{|T|}\}$, to a set of tasks: $T = \{t_1, t_2, \dots, t_{|R|}\}$. A task t is described by a tuple:

$$\langle c_t, dur_t, s_t, [bt_t, et_t], l_t \rangle$$

Where (c_t) is a predefined priority which determines its importance to the company. The higher the value of c_t the more important the task will be, and $c_t \in \mathbb{R}$. dur_t is the expected duration a technician requires to finish this task. Each task requires a technician with a particular skill $s_t \in S$, where S is the set of all skills: $S = \{s_1, s_2, \dots, s_{|S|}\}$. A task t must be serviced within a predefined time window described by $[bt_t, et_t]$. Tasks are geographically distributed, and the location of a task is denoted by l_t .

Each technician $r \in R$ is described by the following tuple:

$$\langle [br_r, er_r], S_r, \ell_r \rangle$$

Each technician has a fixed shift hours where the beginning and end of the shift is expressed by $[br_r, er_r]$. S_r denotes the skill(s) a technician has, where $S_r \subseteq S$. There is no a single depot for technicians to start from, as they can start from home or from one of predefined depots. The location of a technician is denoted by ℓ_r .

The travelling time between any two locations ($TD_{\ell_1 \ell_2}$) is calculated as the Euclidean distance divided by speed (v).

There are two sets of decision variables in *WSP*: the allocation variables $X = \{x_{ij} | i = 1, \dots, |T|; j = 1, \dots, |R|\}$ and the service times $F = \{f_i | i = 1, \dots, |R|\}$. A variable x_{ij} is equal 1 if technician r_i is allocated to task t_j , and 0 otherwise. A variable f_i denotes the start time of service for task t_i .

Having decided these variables, a set of routes $\pi = \{\pi_1, \pi_2, \dots, \pi_{|T|}\}$ are defined. A route π_i is a sequence of tasks ($\subseteq T$) that are to be visited by technician r_i ; $\pi_i = (\pi_{i1}, \dots, \pi_{i\sigma_i}), 1 \leq \sigma_i \leq |R|$.

The main objective is to find an assignment of resources to tasks that maximizes the number of allocated tasks with respect to their priorities, while satisfying all assignment and routing constraints. This paper considers this objective for the *WSP*. There are other objectives which can be considered as well, such as minimizing the total traveling distance, and load balancing.

The command-based Workforce Scheduling Problem can, then, be mathematically modeled as follow:

(*command-based WSP*)

$$\max \sum_{r \in R} \sum_{t \in T} c_t x_{rt} \quad (1)$$

subject to

$$\sum_{r \in R} x_{rt} \leq 1 \quad \forall t \in T \quad (2)$$

$$s_t \in S_r \quad \forall x_{rt} = 1, t \in T, r \in R \quad (3)$$

$$bt_t \leq f_t \quad \forall t \in T \quad (4)$$

$$f_t + dur_t \leq et_t \quad \forall t \in T \quad (5)$$

$$br_r \leq f_{\pi_{ri}} \quad \forall r \in R, i = \{1.. \sigma_r\} \quad (6)$$

$$f_{\pi_{ri}} + dur_{\pi_{ri}} \leq er_r \quad \forall r \in R, i = \{1.. \sigma_r\} \quad (7)$$

$$br_r + TD_{\ell_r \ell_{\pi_{r1}}} \leq f_{\pi_{r1}} \quad \forall r \in R \quad (8)$$

$$f_{t_i} + dur_{t_i} + TD_{\ell_{t_i} \ell_{t_{i+1}}} \leq f_{t_{i+1}} \quad \forall t_i \in \pi_r, \pi_r \in \pi \quad (9)$$

The objective function is represented by (1). Constraint (2) imposes that each task is visited at most once. The skill constraint is expressed in (3). The time-window constraints of all tasks are assured by (4) and (5). (6) and (7) ensure that all tasks which are assigned to a technician must be within the technician's working time. Finally, constraints (8) and (9) ensure route validity by considering the travelling time between a technician's base location and the first task, as well as between subsequent tasks in technician's route.

3 Empowerment

3.1 What does empowerment actually mean?

Empowerment as a management concept is an elastic term which is loosely defined and used [9] [2]. Although it has been widespread since the 1980s, there is no precise definition of empowerment in the literature. Essentially, employee empowerment is a management concept through which employees are given more freedom and flexibility to make decisions related to their work.

Two conceptions of empowerment are reflected in the literature [4] [2] [10] [11]. The first approach conceptualizes empowerment as a managerial relation, and defines this term as the process of enhancing employee authority and control over decisions related to their tasks. From this perspective, empowerment is a broad concept which encompasses other management ideas such as delegation, job enrichment, decentralization of decision making and participatory management. The second approach, however, emphasizes the psychological values of empowerment, and refers to empowerment as the process of enhancing the motivational concept of self-efficacy. Other researchers tend to combine the two approaches, viewing empowerment as enhancing both employee decision power and self-efficacy.

The moralistic theme beneath empowerment language was discussed by Claydon and Doyle [3]. They ethically examined the two messages comprised in the term empowerment, namely the "soft" message (i.e. enhancing autonomy and self-discretion) and the "hard" message (i.e. accountability and ownership). They argue that empowerment language is found between the ethical theories of deontology and egoism.

3.2 Benefits of Empowerment

The empowerment literature delivers convincing arguments concerning the benefits of employee empowerment [12]. Empowerment is seen to be beneficial both for the organization and employees. As Greasley et al (2005) summarizes, the benefits for an organization are the remarkable improvements in cost control, flexibility, productivity and quality; where the benefits of empowered employees are enhanced job satisfaction, motivation and organizational loyalty. These mutual benefits present a win-win scenario which is considered the main motivation for making a move towards empowerment [3].

The literature is rich with success stories from organizations that have embarked on this journey (see for example, [13] [14] [11] [12]). However, such appealing benefits can not be attained without costs, both in terms of establishing a new approach to management (involving training costs, costs of new reward and information systems) and in its operation (involving issues of integration, consistency and unintended consequences) [9].

3.3 Employee's Perception of Empowerment

Researchers have predominantly focused on the management perspective while studying empowerment. However some research has been conducted to study the employee perception of empowerment, for instance [2]. Wilkinson [9] stated that "It is taken for granted in much of the prescriptive literature that employees will welcome and indeed be committed to the new approach. Indeed there is evidence that workers welcome the removal of irritants (e.g. close supervision) and welcome the opportunity to address problems at source as well as the ability to decide work allocation." Yet it has been argued that employee's perception and commitment varies according to several factors such as education, experience, skilfulness, and personal characteristics [10] [14].

3.4 Review of the Empowerment Literature

Most empowerment practices are principally human-centric, in which the amount of redistributed power, alongside the exercises and effectiveness of the empowerment strategy depend entirely on the people in the organization, i.e. managers and employees. It is possible to say that the contribution of technology to this management style is still modest. The main reason behind this is that the empowerment practices suggested in the literature focus on enhancing employees' power and control over task-related decisions, but have not to any great extent been extended to include decisions made by supportive subsystems, particularly the task-assignment system.

Efficient extensions to the scope of practices associated to empowerment to include such decision-support subsystems would, on one hand, create more opportunities for employees to get involved. On the other hand, it could be beneficial for those supportive systems to apply new decision-making strategies that could help improving the service quality.

3.5 Empowerment initiatives in resource scheduling

Before reviewing the models in resource scheduling literature that made an attempt to apply the empowerment management strategy, it is important to define what constitutes empowerment in workforce scheduling problems. We define this as an empowerment-based model should recognize the individuals' self interest, which is somehow reflected in the final plan.

There have been few attempts to implement the empowerment approach in computerized decision support systems in general, and workforce scheduling in particular. It has been reported that an online voting system was used as an empowerment practice, through which employees are given a set of activities and allowed to vote online for the activities they most wish to do [13]. Although voting systems are normally designed to support industrial democracy, it can also be considered as a supportive practice to involve workers in the decision-making process. Nevertheless, this technique is suitable for teamwork, rather than individual, context where democracy could be a very practical practice. Moreover, task assignment problems usually deal with very short-term, partially dynamic jobs, in which there is no sufficient time to utilize this technique.

Employee preferences are considered in several models which were proposed to solve scheduling problems such as employee tour scheduling [15] [16] (for detailed survey see [17]), and task assignment [18] [19]. These models either incorporate information about the preferred working patterns of employees, or allow employees to express their preferences by associating weights to a particular property of the scheduled item e.g. tasks or shifts. Although one could claim that employee preferences is an empowerment practice, it can be considered as a consideration more than involvement, since employees have no explicit authority or control over the decision making. Furthermore, the scope of preference is normally limited to a single attribute of the scheduled item such as the type or location of tasks in task assignment, which could weaken employees' feelings of power and motivation, and consequently their self-efficacy.

The first explicit initiative that adopted empowerment in solving workforce scheduling was introduced by Tsang et al [5]. Their approach was to model the problem's entities (e.g. manager, engineers and jobs) with intelligent agents, each of which has its own interests which are in conflict with others. This approach formulates the problem as a distributed scheduling problem, allowing each agent to look at its interests, while the manager agent looks at the company's interest. Giving engineers the chance to pursue their interests is claimed to be the empowerment practice in this model.

However, employees' power to attain their interests are strongly controlled by the manager agent who generates the weights of engineers' objectives. A schedule is generated as a response to the manager's decision of the objectives' weights; hence, engineers can not anticipate their plans. Furthermore, the difficulty of the manager's optimization problem increases with the number of agents. Thus, implementing individual agents would impact upon the manager's optimality, make this model more appropriately to be implemented at the dispatcher's or probably team leaders' levels, than the individuals. Therefore, this model cannot fully satisfy the constitution of empowerment.

In our research, we intend to develop a model that implements the empowerment management style. The model focuses on formulating the problem as to give employees an explicit control over the decision making. The following section describes our proposed model in detail.

4 Empowerment-based Workforce Scheduling System

Empowerment in practice is a form of employee involvement designed by management with the purpose of boosting workers' contribution and commitment to the organization [9]. Applying the empowerment management approach to workforce scheduling requires designing a flexible, efficient model that increases employee involvement in the allocation process.

In order to deploy the empowerment management concept in the workforce scheduling problem, employees should be involved in the scheduling process and their control over their work should be increased. This can be attained by enabling them to express their preferences and submit daily plans that to be considered in the allocation process.

From the optimization perspective, this involvement would, theoretically, negatively affect, to various extents, the scheduling optimality with respect to the organization's objectives. However, the improvement in service quality and productivity which are promised by the empowerment style motivates the implementation of this management concept.

4.1 Work Plans: The Empowerment Vehicle

A key element in any empowerment-based scheduling system is the representation of employee involvement, which defines the employee decision power. In workforce scheduling problems, employees would like to be empowered to have a decision power that allows them to influence the assignment of jobs to them. If one thinks about the possible scenarios in which an employee would use the power, one could categorize these scenarios into three planning categories:

1. **Short-term plan:** an employee might want to plan for the next job e.g. "I do (not) want to take this particular job due to its requirement or location".
2. **Long-term plan:** an employee might want to plan for a particular day, e.g. "I want to finish my shift at a particular area/location, today I want to be allocated to jobs located in a particular area, or today I want to do only jobs that are of a particular skill(s)".
3. **Preference:** an employee might have preferences he or she would like to be considered in an everyday plan, e.g. "I prefer to work in this area where I usually have family commitments, or I prefer to do jobs that are of a particular skills".

This classification of possible scenarios suggests that the representation of employee involvement in the decision making can be expressed in terms of work plans. A work plan (WP) is a general term which can encompass most possible scenarios. It is the vehicle of empowerment in our model, which employees can utilise to express their own specific preferences.

Naturally all plans can be expressed as a constraint relationship [6]. These constraints will be considered to be satisfied in the scheduling process. Workforce scheduling problems are very complex, subsequently, the promise of satisfying all the plans cannot be guaranteed. Thus an employee's work plan is a soft constraint which can be violated at an incurred cost. In this case the task becomes finding a schedule that satisfies the majority of constraints, rather than all the constraints.

Associating a cost to each plan allows our model to be cope with situations where employees may vary in their power. This variance could be consider as a motivation source for employees, which are normally limited to pay and bonus schemes.

The empowerment practice introduced above changes the nature of the problem. The underlying optimization problem is reformulated to look at not only the organization's objectives, but also to employee interests. Thus, the scheduling problem comprises two sub-problems: an optimization problem that maximizes the organization's objective, subject to the operational constraints, and a constraint satisfaction optimization problem that maximizes satisfied work plans.

4.2 Problem Formulation

Every technician $r \in R$ is able to provide a work plan wp_r per day. A plan is basically a constraint (ρ) by which a technician can describe the jobs he/she wants to undertake. For instance, a technician r might want to be allocated to jobs of a particular set of skills, or jobs in a particular region. Each plan is associated with a weight (ω) to determines the incurred cost of failing to satisfy this plan, in cases where the employees vary in their power over the decision-making process.

An indicator y is defined for the empowerment-based WSP. The indicator y_r ($\forall r \in R$) is equal 1 if the plan ρ_r is satisfied, and 0 otherwise.

As a result, the change this model will make to the formulation of the WSP (section 2.1) is the addition of another objective function; thus, the WSP becomes a bi-objective optimization problem, where the task is not only to maximize the number of jobs served, but also to satisfy the maximum number of technician plans. The mathematical formulation of the empowerment-based WSP will be similar to that of the command-based WSP, with the addition of the following new elements:

$$wp_r = \langle \rho_r, \omega_r \rangle \quad \forall r \in R, \quad 0 \leq \omega_r \leq 1 \quad (10)$$

$$\max \sum_{r \in R} y_r \omega_r \quad (11)$$

In this study, the empowerment-based WSP is dealt with as a single objective optimization problem by combining the the two optimization functions defined in Equations (1) and (11) using weighted-sum approach. The aggregated function is expressed as follows:

$$f(s) = w_1 * \sum_{r \in R} \sum_{t \in T} c_t x_{rt} + w_2 * \sum_{r \in R} y_r \omega_r \quad (12)$$

Where w_1 and w_2 are predefined constants that determine the importance given by the organization to each objective.

5 Prototype

For the model outlined above, we have implemented an initial prototype in which the complexity of the model is slightly reduced. Referring to the formulation of the empowerment-based WSP, the initial prototype assumes that all employees are of an equal power (i.e. all plans are of the same weight). We intend to always examine the tightest scenario, where every employee has a plan to be considered.

5.1 Plan Generator

A major element in our experiment design is to model employee plans. A plan is a constraint that limits the possible values a variable can take. Employees want to be able to describe jobs that they want to be allocated. There are three main properties that can be used to describe a job, namely the job's id, type (i.e. required skill), or location.

Our model is based on these properties, such that each plan uses a property to describe the tasks that the corresponding employee wants to do. When an employee wants to do particular tasks or tasks of a particular skill(s), the employee just needs to determine the task id or the skill(s) codes, respectively. In terms of location, the whole service region of the WSP is clustered into several areas which can be used by employees to limit the jobs that will be assigned to them to particular area(s).

Given the sets of technicians R , tasks T , skills S and areas $Areas$, and the number of tasks, on average, a technician can perform on a daily shift avg_{tr} , the plan generator is outlined as follows:

```

Plans Generator( $R, T, S, Areas, avg_{tr}$ )
1: for each  $r_i$  such that  $1 \leq i \leq |T|$  do
2:    $P \leftarrow \text{Random}(\text{"id"}, \text{"skill"}, \text{"location"})$ 
3:   if  $P = \text{"id"}$  then
4:      $size \leftarrow \text{Random}(avg_{tr} \dots 2 * avg_{tr})$ 
5:      $D_p = \text{Random}(d | d \subseteq T, |d| = size)$ 
6:   else if  $P = \text{"skill"}$  then
7:      $size \leftarrow \text{Random}(1 \dots \max(|S_{r_i}|, |S|/2))$ 
8:      $D_p = \text{Random}(d | d \subseteq S_{r_i}, |d| = size)$ 
9:   else if  $P = \text{"location"}$  then
10:     $size \leftarrow \text{Random}(1 \dots |Areas|/2)$ 
11:     $D_p = \text{Random}(d | d \subseteq Areas, |d| = size)$ 
12:   end if
13: end for

```

The plan of an employee is generated randomly. First, a property will be chosen randomly, which can take three possible values: "id", "skill" or "location", which makes the plan describes tasks by their identifier, skill requirement or location, respectively. Accordingly, the size of the domain values is drawn randomly within a specific range. This range is calculated as a function of the number of tasks, on average, a technician can perform on a daily shift for the "id", the size of the employee's set of skills for "skill", or the number of areas that construct the whole service region for "location". Finally, a subset of the relevant set to the chosen property is randomly selected as the domain values.

Table 1 The probability distribution used to sample the number of skills for each technician.

Number of Skills:	1	2	3	4	5	6	7	8	9	10
Probability:	0.02	0.05	0.08	0.15	0.20	0.20	0.15	0.08	0.05	0.02

5.2 Problem instances

We developed a problem generator which is partially inspired by a real WSP. Instances are constructed as follows: We measure the duration of a day in the number of minutes from midnight. A day typically starts at 480 (8:00 am) and ends at 1060 (5:40 pm), i.e. $[br_r, er_r] = [480, 1060]$. We assume that all technicians have only a full-day shift, to minimize noise.

In a real scenario, a region manages 86 technicians ($|R| = 86$), and receives on average about 260 tasks per day ($|T| \simeq 260$), of which 222 are appointment (i.e. normal priority, $c_{np} = 10$) tasks. For appointment tasks, we distinguish between a whole day task (which can be done anytime during the day) a morning task which should be finished by 13:00 o'clock and an evening tasks which cannot be started before 13:00. In the experiment, we set the time window of a task t , $[bt_t, et_t]$, to be $[480, 1060]$ with 0.5 probability, $[480, 780]$ with 0.25 probability and $[780, 1060]$ with 0.25 probability. On the other hand, high priority tasks ($c_{hp} = 20$) start differently during the day, all of which must be served within 180 from the time it starts (i.e $et - bt = 180$).

Each task requires a technician with a particular skill. We distinguish between ten different skills ($|S| = 10$), abbreviated: $s_1 \dots s_{10}$. Each technician can have one or more skills. In our experiment, the number of skills that technicians in the workforce have (denoted by μ) follows a binomial-like distribution as given in Table 1. The type of skills for both tasks and technicians is sampled uniformly at random from $s_1 \dots s_{10}$.

The time required to finish a task (dur_t) varies as a function of the associated skill. This is modelled as a triangular distribution. However, we consider in this paper a simple scenario, in which the duration of all tasks, by default, is equal to 95.

We consider a 80 km^2 region. The density of tasks over this region is based on real geographical distribution of houses. We assume that technicians start the working day from home. The location of all tasks as well as the initial locations of all technicians are chosen uniformly at random from the region. Travel time (between any two coordinates) is measured as the Euclidean distance divided by speed (v). We are particularly interested in highly constrained travel times and hence we assume that the speed (v) is 5 km/h.

5.3 Algorithm

The aim of the workforce scheduling algorithm is to find a feasible schedule of high-quality with respect to the main objective. However, most of the real-world scheduling problems are combinatorial optimization problem in which using complete search methods (e.g. branch & bound method) to find the optimal solution in a reasonable time is not possible. Thus, it is usually the case that one sacrifices completeness and look for a near-optimal solution at reasonable time.

Two major approaches which use this strategy have been successfully applied to workforce scheduling problems, namely metaheuristics [20] such as Guided Local Search

[19], the Genetic Algorithm [21], Simulated Annealing [22] and Constraint Logic Programming [23,24]; and rule-based systems [25,26]. Among these approaches, Guided Local Search obtained the best results on a benchmark problem [19], which motivates us to study the application of Guided Local Search to the empowerment-based WSP.

5.3.1 The Basic Local Search

Local Search (LS) is the basis of most heuristic methods including Guided Local Search [20]. Follows are the four main components need to be defined to apply LS to the empowerment-based WSP:

Solution Representation. Each candidate solution, i.e. state, in the search space is represented by a permutation of $|T|$ tasks. Such a permutation specifies the order of tasks to be considered in the scheduling process. The first task in the permutation is scheduled first, then the second one, and so on and so forth. Scheduling a particular task follows a deterministic procedure which can be summarized as allocate this task to the nearest technician with respect to all constraints and employee plans.

Neighbourhood Function. The Neighbourhood function is defined as any new permutation that may be obtained from the current permutation by a single swap between any two tasks.

Cost Function. For the empowerment-based WSP, the cost function ($f(s)$) will be the weighted-sum function defined in Equation (12).

The initial solution. The initial solution is generated heuristically based on three rules:

1. High priority tasks are scheduled first. This rule is very practical since the objective function takes into consideration the priorities of tasks (c_t).
2. For two tasks t_i, t_j with the same priority (i.e. $c_{t_i} = c_{t_j}$), if the number of available and qualified engineers for t_i is less than that for t_j , then t_i is scheduled first, otherwise t_j (i.e. Smallest-domain-first, SDF).
3. For two tasks t_i, t_j with the same priority and domain size, the shorter in terms of estimated duration dur_t is scheduled first (i.e. greedy principle).

5.3.2 Guided Local Search

Guided Local Search (GLS) is a general metaheuristic algorithm for optimisation [27]. GLS sets on top of local search with the purpose of escaping from local minima by using a penalty-based approach. The basic idea of GLS is to penalize selected features of local optima. Therefore, GLS principle is based on defining solution features (i.e. attributes that can be used to discriminate between different solutions). Instead of the cost function $f(s)$, the LS searches in the space of solutions by using $f(s)$ augmented by the accumulated penalties:

$$h(s) = f(s) + \lambda \sum_{i \in F} p_i * I_i(s) \quad (13)$$

In this formula, s is a candidate solution and λ is a parameter to the GLS algorithm. F refers to the defined feature set in s . p_i is the penalty of feature i (each p_i is initialized to 0), and I_i is an indicator: $I_i(s) = 1$ if s exhibits the feature i ; 0 otherwise.

The novelty of GLS is in the approach it deploys to define the bad features to penalize. This definition is based on two factors, namely feature cost and the frequency

of penalizing a feature. The utility of penalizing feature i , $util_i$, under a local optimum s^* , is defined as follows:

$$util_i(s^*) = I_i(s^*) \times \frac{c_i}{1 + p_i} \quad (14)$$

Where c_i is the cost and p_i is the current penalty value of feature i .

In the empowerment-based WSP, $f(s)$ is an aggregation of two functions with different weights and, thus, two sets of features will be defined. As minimizing the total unallocated tasks is an objective, the inability to serve a task is characterized as a feature to be penalized. Thus, the first set of features F_1 has $|T|$ features, where $|T|$ is the total number of tasks. The second objective concerns minimizing unsatisfied employees' plans and, therefore, the second set of features F_2 are characterized by the inability to satisfy plans. There are $|R|$ features in F_2 , where $|R|$ is the total number of employees.

Each feature is associated with a cost to help GLS choosing features that have more influence on the cost function in order to penalize them. We consider the task priority as the cost of features in F_1 , in order to give high priority tasks more importance; whereas all features in F_2 are of an equal cost, due to the assumption that all employees have the similar power. p_i is the penalty of feature i (each p_i is initialized to 0), and I_i is an indicator: $I_i(s) = 1$ if s exhibits feature i ; 0 otherwise.

As a result, the augmented objective function $h(s)$ is expressed as follows:

$$h(s) = w_1 * [\sum_{r \in R} \sum_{t \in T} c_t x_{rt} + \lambda_1 \sum_{i \in F_1} p_i * I_i] + w_2 * [\sum_{r \in R} y_r \omega_r + \lambda_2 \sum_{i \in F_2} p_i * I_i] \quad (15)$$

The weighted-sum method applied to transfer the two objectives functions into a single function, is also propagated to the corresponding set of features. We, moreover, defined two parameters of the GLS λ_1 and λ_2 ; however, the empowerment-based WSP was found, through empirical analysis, not to be sensitive to those parameters.

6 Results and Discussion

In this paper, we aim to examine the new model, in comparison to the command-based model. This includes examining the cost of different level of employee power over the scheduling decision on the company's main objective (i.e. productivity objective), and the ability of this model to provide an efficient empowerment practice.

To attain this aim, we ran the GLS algorithm on 20 problem instances, for each of which different sets of weights (w_1, w_2) were tested, such that $(w_1, w_2) \in \{ (1,0), (0.8,0.2), (0.6,0.4), (0.5,0.5), (0.4,0.6), (0.2,0.8), (0,1) \}$. At one extreme, $w_1 = 1$ represents the command-based WSP, since the empowerment objective (Equation 10) is ignored in the optimization process. And as w_1 decreases, the influence of the empowerment objective on the optimization process should increase.

Since the plans generator is based on a stochastic model, five different instances of employees' plans were tested for each weight setting.

In order to analyse the influence of empowerment on the productivity objective, we use the actual cost of each objective rather than the aggregation of both objectives. For each weight setting, the mean of five runs with different employee plans is calculated for each instance and, then, the mean of 20 instances is measured.

The results are given in Table 2 which describes the mean cost of each objective for each weight setting. For simplicity, the productivity loss is included. The correlation

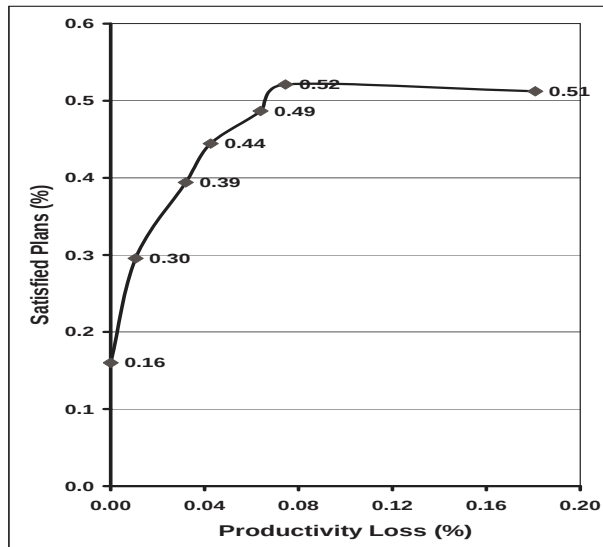


Fig. 1 The correlation between the empowerment objective and the productivity objective.

between the productivity loss and the empowerment cost is plotted in Fig1, which shows the cost of satisfying employee plans on the company's main objective.

The results reveal the efficiency of this model which shows its ability to consider the plans from all employees and satisfy 52% of them. This is achieved at the expense of losing 7% of the productivity objective. The inability to satisfy all plans is expected and this is due to either the existence of conflict between multiple plans, in which only one of them can be satisfied, or to the inability of the optimization algorithm to obtain the global optimum solution.

Furthermore, the results encouragingly show that even if the empowerment objective is ignored (i.e. the traditional WSP); about 16% of all plans can be satisfied without any effect on the productivity objective.

Interestingly, when the GLS optimizes only the empowerment objective, it could not obtain the best cost with respect to this objective. This can be attributed to the many local minima in the landscape of the empowerment objective which traps the GLS. To

Table 2 The mean cost of each objective for each weight setting

Weight Setting (Pro,Emp)	Productivity (Pro%)	Productivity Loss (%)	Satisfied Plans(Emp%)
(1,0)	0.94	0	0.16
(0.8,0.2)	0.93	0.01	0.30
(0.6,0.4)	0.91	0.03	0.39
(0.5,0.5)	0.9	0.04	0.44
(0.4,0.6)	0.88	0.06	0.49
(0.2,0.8)	0.87	0.07	0.52
(0,1)	0.77	0.18	0.51

illustrate this, the size of the search space for the problems used in the experiments is about 260!, the domain of values for the cost of each of which range from $0 \cdots |R| (= 86)$. Therefore, introducing another objective with a small weight to be aggregated changes the landscape and helps the GLS to explore more spaces.

7 Conclusions and Future Work

The Workforce Scheduling Problem (WSP) is a complex combinatorial optimization problem. Employee efficiency is very critical to the effectiveness of any scheduling system; however, conventional formulations of WSPs exclude employees from the decision-making process.

Empowerment is an alternative management concept, through which employees are given more control over decisions related to their work. Enhancing employee efficiency is a major benefit promised by empowerment, which motivates us to incorporate this management theme in designing the WSP.

The empowerment-based WSP is a new formulation of the WSP, which implements the empowerment management concept by incorporating ideas from the constraint satisfaction discipline. This formulation considers the two constitutions of empowerment in WSPs, namely recognizing individuals self interest and enhancing individuals' feeling of empowerment which can be attained when their plans are reflected in the final schedule.

Based on this new formulation, a prototype has been developed to prove the concept that the formulation satisfies the two constitutions of empowerment. A Guided Local Search was proposed to solve the underlying optimization problem in this prototype. Experiments were conducted to analyse the efficiency of this new approach, in comparison to the traditional formulation. The results showed the efficiency of this model, demonstrated by the ability of GLS to satisfy about 52% of all employee plans at the expense of sacrificing 7% of the company's main objective.

The empowerment-based WSP is a constrained multi-objective optimization problem in nature. In this study, we rather transformed the problem to a single objective optimization problem by using the weighted-sum approach. However, the multi-objective optimization literature is rich with efficient techniques that can be applied to the empowerment-based WSP.

We developed a stochastic model to simulate employee plans. We intend to investigate other stochastic or heuristic models in order to examine the robustness of this formulation, alongside any proposed algorithm, in relation to changes in employee plans.

References

1. Voudouris, C., Owusu, G., Dorne, R., Lesaint, D.: Service Chain Management: Technology Innovation for the Service Business. Springer Berlin Heidelberg (2008)
2. Greasley, K., Bryman, A., Dainty, A., Price, A., Soetanto, R., King, N.: Employee perceptions of empowerment. *Employee Relations* **27**, 354–368
3. Claydon, T., Doyle, M.: Trusting me, trusting you? the ethics of employee empowerment. *Personnel Review* **25**, 13–25
4. Conger, J.A., Kanungo, R.N.: The empowerment process: Integrating theory and practice. *The Academy of Management Review* **13**(3), 471–482 (1988)

5. Tsang, E.P.K., Virginas, B., Gosling, T., Liu, W.: Multi-agent systems for staff empowerment. In: *Service Chain Management*, pp. 263–272
6. Tsang, E.: *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego (1993)
7. Borrett, J., Tsang, E.: A context for constraint satisfaction problem formulation selection. *Constraints* **6**(4), 299–327 (2001). 593341
8. Fink, E.: How far solve it automatically: Selection among problem-solving methods. In: *4th Int. Conference on Artificial Intelligence Planning Systems* (1998)
9. Wilkinson, A.: Empowerment: theory and practice. *Personnel Review* **27**, 40–56
10. Dimitriadis, Z.S.: Empowerment in total quality: Designing and implementing effective employee decision-making strategies. *Quality Management Journal* **8**(2), 19–28 (2001)
11. Ahearne M., M.J.R.A.: To empower or not to empower your sales force? an empirical examination of the influence of leadership empowerment behaviour on customer satisfaction and performance. *Journal of Applied Psychology* **90**(5), 945–955 (2005)
12. Hales, D.: How far dare you go? *Stakeholder* **2**(4), 31–34 (2005)
13. Silver, S., Randolph, W.A., Seibert, S.: Implementing and sustaining empowerment: Lessons learned from comparison of a for-profit and a nonprofit organization. *Journal of Management Inquiry* **15**(1), 47–58 (2006)
14. Hirschheim, R., Miller, J.: Implementing empowerment through teams: the case of texaco's information technology division. In: *SIGCPR '93: Proceedings of the 1993 conference on Computer personnel research* (1993)
15. Love, R., Hoey, J.: Management science improves fast-food operations. *Interfaces* **20**(2), 21–29 (1990)
16. Yura, K.: Production scheduling to satisfy workers preferences for days off and overtime under due-date constraints. *International Journal of Production Economics* **33**, 265–270 (1994)
17. Alfares, H.: Survey, categorization, and comparison of recent tour scheduling literature. *Annals of Operations Research* **127**, 145–175 (2004)
18. Tsang, E., Gosling, T., Virginas, B., Voudouris, C., Owusu, G.: Retractable contract network for distributed scheduling. In: *2nd Multidisciplinary International Conference on Scheduling: Theory & Applications (MISTA)*, pp. 485–500. New York (2005)
19. Tsang, E., Voudouris, C.: Fast local search and guided local search and their application to british telecom's workforce scheduling problem. *Operations Research Letters, Elsevier Science Publishers, Amsterdam* **20**(3), 119–127 (1997)
20. Glover, F., Kochenberger, G.: *Handbook of Metaheuristics* (International Series in Operations Research & Management Science). Springer (2003)
21. Muller, C., Magill, E., Smith, D.: Distributed genetic algorithms for resource allocation. Technical report, Strathclyde University (1993)
22. Baker, S.: Applying simulated annealing to the workforce management problem. Technical report, British Telecom Laboratories (1993)
23. Lever, J., Wallace, M., Richards, B.: Constraint logic programming for scheduling and planning. *British Telecom Technology Journal* **13**(No.1), 73–80 (1995)
24. Yang, R.: Solving a workforce management problem with constraint programming. In: *The 2nd International Conference on the Practical Application of Constraint Technology*, pp. 373–387 (1996)
25. Jackson, P.: *Introduction to Expert Systems*, 3rd edn. Addison Wesley Longman, Harlow, England (1999)
26. Abraham, A.: Rule based expert systems. In: P.S. Thorn, Richard (eds.) *Handbook for Measurement Systems Design*, pp. 909–919. John Wiley and Sons Ltd., London (2005)
27. Voudouris, C., Tsang, E.: Guided local search. In: F. Glover (ed.) *Handbook of metaheuristics*, pp. 185–218. Kluwer (2003)

Master Physician Scheduling Problem

Aldy Gunawan • Hoong Chuin Lau

Abstract In this paper, we study a real-world problem arising from the operations of a hospital service provider, which we term the master physician scheduling problem. It is a tactical planning problem of assigning physician duties to the defined time slots/shifts over a time horizon incorporating a large number of constraints and complex physician preferences. The goals are to satisfy as many physicians' preferences and duty requirements as possible while ensuring optimum usage of available resources. We propose mathematical programming models that represent different variants of this problem. The developed models were tested on a real case from the Surgery Department of a large local government hospital, as well as on randomly generated problem instances. The computational results for solving the model are reported together with some comparison and analysis of the optimal solutions obtained. We show that this approach can significantly reduce the time and effort required to construct the master physician scheduling schedule.

Keywords: master physician scheduling and rostering problem, mathematical programming, optimization, preferences.

1 Introduction

Recently, there has been a greatly increased interest in hospital operations management which involves optimized assignment of individuals including physicians, nurses and administrators. One of the complex tasks in hospital management is to design a physician schedule, which requires taking into account a large number of constraints and preferences.

A physician schedule provides an assignment of physicians to perform different duties in the hospital timetable. Unlike nurse rostering problems which have been extensively studied in the literature (e.g. Ernst et al, 2004; Beliën and Demeulemeester, 2005; Petrovic and Berghe, 2008), in physician scheduling, maximizing satisfaction only matters, as physician retention is the most critical issue faced by hospital administrations (Carter and Lapierre, 2001). In addition, while nurse schedules must adhere to collective union agreements, physician schedules are more driven by personal preferences. In general, planning the schedules for

Aldy Gunawan

Research Fellow, The Logistics Institute – Asia Pacific, National University of Singapore
7 Engineering Drive 1, Singapore 117574

E-mail: tliag@nus.edu.sg

Hoong Chuin Lau

Associate Professor, School of Information Systems, Singapore Management University
80 Stamford Road, Singapore 178902

E-mail: hclau@smu.edu.sg

physicians requires satisfying a very large number of (often conflicting) constraints and preferences. Carter and Lapierre (2001) also provides the fundamental differences between physicians and nurses scheduling problems.

To our knowledge, research on physician scheduling focuses primarily on a single type of duty, such as the emergency room (e.g. Vassilacopoulos, 1985; Beaulieu et al., 2000; Carter and Lapierre, 2001; Gendreau et al., 2007), the operating room (e.g. Testi et al., 2007; Burke and Riise, 2008), the physiotherapy and rehabilitation services (Ogulata et al., 2008).

In this paper, we consider the problem of generating a master schedule for the physicians within a hospital service by taking a full range of day-to-day duties/activities of the physicians (including surgery, clinics, scopes, calls, administration) into consideration. Our problem, termed the **Master Physician Scheduling Problem**, is the tactical planning problem of assigning physician activities to the time slots over a time horizon incorporating a large number of rostering and resource constraints together with complex physician preferences. The goals are to satisfy as many physicians' preferences and duty requirements as possible while ensuring optimum usage of available resources such as clinics and operating theatres.

The major contributions/highlights of this paper are as follows:

- (1) We take a physician-centric approach to solving this problem, since physician retention is the most critical issue faced by hospital administrations worldwide.
- (2) Using mathematical models, we provide a comprehensive empirical understanding of the tradeoff of constraints and preferences against resource capacities.

The paper is organized as follows. Section 2 provides a review of the literature. Section 3 gives a detailed description of the problem. The mathematical programming models are presented in Section 4. Our developed models are tested on a real case from the Surgery Department of a large local government hospital, as well as on randomly generated problem instances. Computational results are reported together with our analysis in Section 5. Finally, we provide some concluding perspectives and directions for future research in Section 6.

2 Literature Review

Personnel scheduling and rostering is the process of constructing optimized work timetables for staff in an organization, and is becoming a critical concern in service organizations such as hospitals, hotels and airlines. A number of reviews in personnel scheduling and rostering research have appeared in Aggarwal (1982), Burke et al. (2004), Ernst et al. (2004). A categorization of comprehensive and representative solution techniques employed for different rostering problems are found in Ernst et al. (2004).

Beaulieu et al. (2000) claimed that their work was the first to present a mathematical programming approach for scheduling physicians in the emergency room. The proposed model is based on multi-objective integer programming theory. The constraints are partitioned into four different categories according to the types of rules to which they correspond: compulsory constraints, ergonomic constraints, distribution constraints, and goal constraints.

Gendreau et al. (2007) presented several generic forms of the constraints encountered in six different hospitals in the Montréal area (Canada) as well as several possible solution techniques for solving the problem. The constraints of the physician scheduling problem can be classified into four categories: supply and demand, workload, fairness and ergonomic constraints. In this paper, we are concerned about the physicians' preferences instead of fairness constraints.

A number of exact and heuristic algorithms for various scheduling problems encountered in hospitals were summarized by Beliën (2007). Beaulieu et al. (2000) proposed a mixed 0-1 programming formulation of the physician scheduling problem. The objective function is the sum of deviation (penalty) of some constraints. The problem was then solved by a heuristic approach based on a partial branch-and-bound. Buzon and Lapierre (1999) applied Tabu Search to the acyclic schedules. The cost of the solution is the sum of the costs of all physician schedules, which each one represents the sum of all penalties associated with the unsatisfied constraints.

Rousseau et al. (2002) and Bourdais et al. (2003) applied constraint programming to the physician scheduling problem. This method has been applied to the nurse scheduling problem (Bard and Purnomo, 2005). The solution technique can also be applied to the physician scheduling problem after some minor modifications.

3 Problem Description

The problem addressed in this study is to assign different physician duties (or activities) to the defined time slots over a time horizon incorporating a large number of constraints and complex physician preferences. In this paper, we assume the time horizon to be one work week (Mon-Fri), further partitioned into 5 days and 2 shifts (AM and PM).

Physicians have a fixed set of duties to perform, and they may specify their respective *ideal schedule* in terms of the duties they like to perform on their preferred days and shifts, as well as shifts-off or days off. Taking these preferences together with resource capacity and rostering constraints into consideration, our goal is to generate an *actual schedule*. As shown in Figure 1 as example, the ideal schedules might not be fully satisfied in the actual schedule. That may occur in two scenarios:

- Some duties have to be scheduled on different shifts or days – which we term *non-ideal scheduled duties* (e.g. Physician 2 Tuesday duties).
- Some duties simply cannot be scheduled due to resource constraints – which we term *unscheduled duties* (e.g. Physician 1 Friday PM duty).

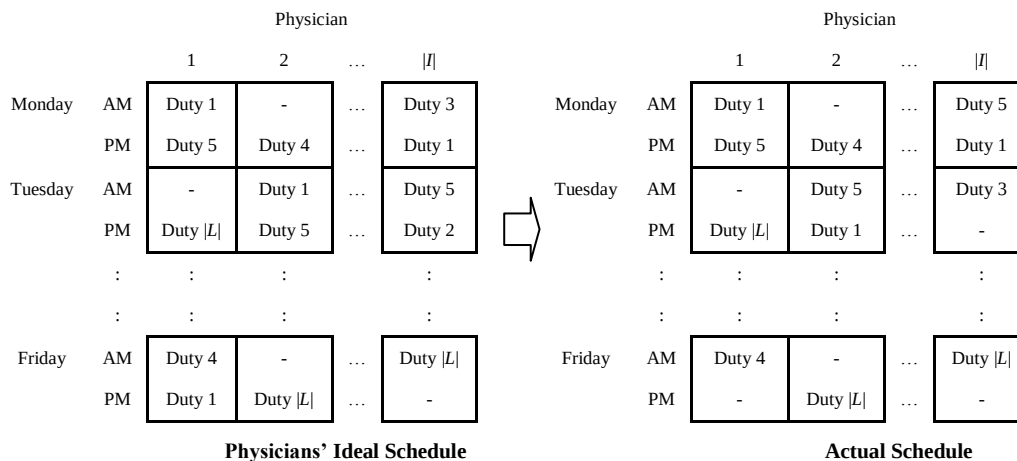


Figure 1. Example of Master Physician Scheduling Problem

Although each hospital has its unique rostering requirements, the following summarizes some common requirements treated in this paper:

- No physician can perform more than one duty in any shift.
- The number of resources (e.g. operating theatres, clinics) needed cannot exceed their respective capacities at any time. For simplicity, we assume that each type of activity does not share its resources with another type of activities – for example, operating theatres and clinics are used to perform surgery and out-patient duties respectively.
- Ergonomic constraints: Some duties are regarded as *heavy* duties, such as surgery and endoscopy duties. The following ergonomic constraints hold:
 - If a physician is assigned to a heavy duty in the morning shift, then he cannot be assigned to another type of heavy duty in the afternoon shift *on the same day*. However, it is possible to assign *the same* type of heavy duties in consecutive shifts on the same day.

- Similarly, a physician cannot also be assigned to another type of heavy duty in the morning shift on a particular day if he has been assigned to a heavy duty in the afternoon shift on the previous day.
- The number of activities allocated to each physician cannot exceed his contractual commitments, and do not conflict with his external commitments. In this paper, we assume external commitments take the form of physicians' request for shifts-off or days-off, and hence no duty should be assigned to these requests.

In this paper, we study different problem settings. The basic problem is to minimize the total number of unscheduled duties without any physician preferences nor ergonomic constraints. From this basic problem, we derive two major problem settings. The first is the problem of satisfying the physicians' ideal schedule as far as possible (or minimizing the total number of non-ideal scheduled duties) while not compromising on having the minimum number of unscheduled duties. The second is the setting where physicians do not provide their ideal schedule, but instead ergonomic constraints are employed across all physicians in minimizing the total number of unscheduled duties.

4 Mathematical Programming Models

The following notations are used in this paper.

Basic notations:

- I = Set of physicians, $i \in \{1, 2, \dots, |I|\}$
- J = Set of days, $j \in \{1, 2, \dots, |J|\}$
- K = Set of shifts per day, $k \in \{1, 2, \dots, |K|\}$
- L = Set of duties, $l \in \{1, 2, \dots, |L|\}$

The following sets are defined to facilitate definition of constraints:

- L^C = $\{l \in L : l = \text{duty with the limited number of resources available}\}$
- $L^{C'}$ = $\{l \in L : l = \text{duty without the limited number of resources available}\}$, while $L^C \cup L^{C'} = L$
- L^H = $\{l \in L : l = \text{heavy duty}\}$
- $PRA = \{(i, j, k) \in I \times J \times K : (i, j, k) = \text{physician } i \text{ requests not being assigned on day } j \text{ shift } k\}$

The following data parameters are used:

- R_l = number of resources required to perform duty l ($l \in L^C$)
- C_{jkl} = number of resources available for duty l on day j shift k ($j \in J, k \in K, l \in L^C$)
(i.e. resource capacity)
- A_{il} = number of duty l requested by physician i in a weekly schedule ($i \in I, l \in L$)
- F_{ijkl} = 1 if physician i requests duty l on day j shift k , 0 otherwise

The decision and auxiliary variables are used:

- X_{ijkl} = 1 if physician i is assigned to duty l on day j shift k , 0 otherwise
- d_{ijkl}, d'_{ijkl} = deviations between the actual and ideal schedules of physician i to duty l on day j shift k ($d_{ijkl} \geq 0, d'_{ijkl} \geq 0$), related as follows:
 - if duty l of physician i can be scheduled on day j shift k as requested in the ideal schedule, then $d_{ijkl} = 0$ and $d'_{ijkl} = 0$ (i.e. no deviation);
 - if duty l of physician i cannot be scheduled on day j shift k as requested in the ideal schedule, then $d_{ijkl} = 1$ and $d'_{ijkl} = 0$;
 - if duty l of physician i is scheduled on another day or shift, then $d_{ijkl} = 0$ and $d'_{ijkl} = 1$.

D_i = sum of deviations of physician i (see equation (17))
 U_i = number of unscheduled duties of physician i
 N_i = number of non-ideal scheduled duties of physician i (for Model IIa only)

Our problem can then be formulated by the following mathematical models. Model I is the base model which computes the minimum number of unscheduled duties for the unconstrained problem. This value provides the upper bound on the total unscheduled duties for subsequent constrained models involving physician preferences (Model IIa) and ergonomic constraints (Model IIb).

[Model I]

$$\text{Minimize } Z = \sum_{i \in I} U_i \quad (1)$$

subject to:

$$R_l \times \sum_{i \in I} X_{ijkl} \leq C_{jl} \quad j \in J, k \in K, l \in L^C \quad (2)$$

$$\sum_{j \in J} \sum_{k \in K} X_{ijkl} \leq A_{il} \quad i \in I, l \in L \quad (3)$$

$$\sum_{l \in L} X_{ijkl} \leq 1 \quad i \in I, j \in J, k \in K \quad (4)$$

$$\sum_{l \in L} X_{ijkl} = 0 \quad (i, j, k) \in PRA \quad (5)$$

$$U_i = \sum_{l \in L} A_{il} - \sum_{j \in J} \sum_{k \in K} \sum_{l \in L} X_{ijkl} \quad i \in I \quad (6)$$

$$X_{ijkl} \in \{0,1\} \quad i \in I, j \in J, k \in K, l \in L \quad (7)$$

$$U_i \in Z^+ \quad i \in I \quad (8)$$

(1) is the total number of unscheduled duties that needs to be minimized. Constraint (2) is the resource capacity constraint (total number of resources required does not exceed total number of available resources per shift). (3) represents the number of duties allocated to each physician cannot exceed his contractual commitments. (4) ensures that each physician cannot be assigned more than one duty in any shift, while (5) ensures that no duty would be assigned to a physician during any shifts-off or days-off requested. (6) defines the number of unscheduled duties (which is to be minimized). (7) imposes the 0-1 restrictions for the decision variables X_{ijkl} , while (8) is the nonnegative integrality constraint for the decision variables U_i .

Model IIa extends the base model considering physicians' ideal schedule. Let U_i^* denote the optimal solution containing the number of unscheduled duties for physician i obtained by Model I. We constrain the number of unscheduled for each physician to not exceed this upper bound (see (10)). Model IIa seeks to then minimize the total number of non-ideal scheduled duties. The deviations between the ideal and the actual schedules for each physician are represented by decision variables d_{ijkl} and d'_{ijkl} , as shown in (16). The total number of non-ideal duties for each physician is calculated by equations (15)–(18). The rest of the constraints are identical to those of Model I.

[Model IIa]

$$\text{Minimize } Z = \sum_{i \in I} N_i \quad (9)$$

subject to:

$$U_i \leq U_i^* \quad i \in I \quad (10)$$

$$R_l \times \sum_{i \in I} X_{ijkl} \leq C_{jl} \quad j \in J, k \in K, l \in L^C \quad (11)$$

$$\sum_{j \in J} \sum_{k \in K} X_{ijkl} \leq A_{il} \quad i \in I, l \in L \quad (12)$$

$$\sum_{l \in L} X_{ijkl} \leq 1 \quad i \in I, j \in J, k \in K \quad (13)$$

$$\sum_{l \in L} X_{ijkl} = 0 \quad (i, j, k) \in PRA \quad (14)$$

$$U_i = \sum_{l \in L} A_{il} - \sum_{j \in J} \sum_{k \in K} \sum_{l \in L} X_{ijkl} \quad i \in I \quad (15)$$

$$X_{ijkl} + d_{ijkl} - d'_{ijkl} = F_{ijkl} \quad i \in I, j \in J, k \in K, l \in L \quad (16)$$

$$D_i = \sum_{j \in J} \sum_{k \in K} \sum_{l \in L} (d_{ijkl} + d'_{ijkl}) \quad i \in I \quad (17)$$

$$N_i = (D_i - U_i)/2 \quad i \in I \quad (18)$$

$$X_{ijkl}, d_{ijkl}, d'_{ijkl} \in \{0,1\} \quad i \in I, j \in J, k \in K, l \in L \quad (19)$$

$$U_i, N_i, D_i \in Z^+ \quad i \in I \quad (20)$$

The second problem setting we consider is where physicians do not express their ideal schedule but instead ergonomic constraints are imposed which aim to improve the quality of the schedule of each physician, formulated as Model IIb. Here, duties are classified into two different groups: *heavy* and *light* duties. A physician assigned to a *heavy* duty in a particular shift cannot be assigned to different *heavy* duties in the next shift on the same day, as represented by (27). If a physician is assigned to a *heavy* duty in the last shift on a particular day, he cannot also be assigned to different *heavy* duties in the first shift on the next day (see (28)). This condition implies that physicians might perform the same heavy duties in consecutive shifts.

[Model IIb]

$$\text{Minimize } Z = \sum_{i \in I} U_i \quad (21)$$

subject to:

$$R_l \times \sum_{i \in I} X_{ijkl} \leq C_{jkl} \quad j \in J, k \in K, l \in L^C \quad (22)$$

$$\sum_{j \in J} \sum_{k \in K} X_{ijkl} \leq A_{il} \quad i \in I, l \in L \quad (23)$$

$$\sum_{l \in L} X_{ijkl} \leq 1 \quad i \in I, j \in J, k \in K \quad (24)$$

$$\sum_{l \in L} X_{ijkl} = 0 \quad (i, j, k) \in PRA \quad (25)$$

$$U_i = \sum_{l \in L} A_{il} - \sum_{j \in J} \sum_{k \in K} \sum_{l \in L} X_{ijkl} \quad i \in I \quad (26)$$

$$X_{ijkl_1} + X_{ij(k+1)l_2} \leq 1 \quad i \in I, j \in J, k \in \{1, 2, \dots, |K|-1\}, l_1 \& l_2 \in L^H (l_1 \neq l_2) \quad (27)$$

$$X_{ij|K|l_1} + X_{i(j+1)l_2} \leq 1 \quad i \in I, j \in \{1, 2, \dots, |J|-1\}, l_1 \& l_2 \in L^H (l_1 \neq l_2) \quad (28)$$

$$X_{ijkl} \in \{0,1\} \quad i \in I, j \in J, k \in K, l \in L \quad (29)$$

$$U_i \in Z^+ \quad i \in I \quad (30)$$

5 Computational Results

In this section, we report a comprehensive suite of experimental results which aims to provide computational perspectives on one hand, and insights to hospital administrators.

5.1 Experimental Setup

First we discuss how problem instances were generated. The 6 sets of random instances were generated with varying values of the following parameters - total percentage of *heavy* duties assigned to physicians (last column of Table 1) and number of resources available in every shift (Table 2). The rationale behind this scenario is to conduct the sensitivity analysis that would be described more detail after the results reported. Other parameters, such as number of *heavy* duties and number of duties with limited resource capacity are set to constant. Besides random instances, we also provide a real case study provided by the Surgery

Department of a large local government hospital. Table 1 summarizes the characteristics of each problem set.

Table 1. Characteristics of Problem Instances

Problem Set	Number of physicians	Number of shifts per day	Number of days	Number of duties	Number of heavy duties	Number of duties with limited capacity	Total percentage of heavy duties*
Case study	15	2	5	9	3	3	73%
Random 1	20	2	5	7	3	3	20%
Random 2	20	2	5	7	3	3	30%
Random 3	20	2	5	7	3	3	40%
Random 4	20	2	5	7	3	3	50%
Random 5	20	2	5	7	3	3	60%
Random 6	20	2	5	7	3	3	70%

$$* = (\sum_{i \in I} \sum_{l \in L^H} A_{il} / |I| \times |J| \times |K|) \times 100\%$$

In order to generate sufficient hard problem instances for the purpose of sensitivity analysis, the resource capacity has to be carefully set, which we explain as follows. The upper bound for number of available resources for each random problem set is set as follows:

$$C_{jkl} = \left\lceil \frac{const \times R_l \times \sum_{i \in I} A_{il}}{|K| \times |J|} \right\rceil \quad j \in J, k \in K, l \in L^C \quad (31)$$

where *const* is a scale factor.

For each problem set, we generate different instances by varying the values of C_{jkl} as follows. For each duty l with the highest total number of resources required, the value of *const*

is set to 2, and the value of C_{jkl} decreases by one unit until it is equal to $\left\lceil \frac{R_l \times \sum_{i \in I} A_{il}}{|K| \times |J|} \right\rceil - 1$

(for $j \in J, k \in K$). For other duties, the value of C_{jkl} is set from $\left\lceil \frac{R_l \times \sum_{i \in I} A_{il}}{|K| \times |J|} \right\rceil + 1$ to

$$\left\lceil \frac{R_l \times \sum_{i \in I} A_{il}}{|K| \times |J|} \right\rceil - 1.$$

Table 2. Examples of varying values of C_{jkl} (Random 1 and Random 2 instances)

Problem Set	Instances	L^C		
		Duty 1	Duty 2	Duty 3
Random 1		15	28	22
	Random 1a	3	6	4
	Random 1b	3	5	4
	Random 1c	3	4	4
	Random 1d	3	3	4
	Random 1e	3	3	3
	Random 1f	2	3	3
	Random 1g	1	2	2
Random 2		21	46	32
	Random 2a	4	10	5
	Random 2b	4	9	5
	Random 2c	4	8	5
	Random 2d	4	7	5
	Random 2e	4	6	5
	Random 2f	4	5	5
	Random 2g	4	5	4
	Random 2h	3	5	4
	Random 2i	2	4	3

As an illustration, Table 2 presents three columns that give the varying values of C_{jkl} for different duties defined in the instances of the Random 1 and Random 2 problem sets. For Random 1, the total number of resources required for Duty 1, Duty 2 and Duty 3 are 15, 28 and

22, respectively. Here, Duty 2 is the duty with the highest number of resources required and hence the value of C_{jkl} for Duty 2 is set to $\left\lceil \frac{2 \times 28}{2 \times 5} \right\rceil$ and decreases by one unit until it is equal to

$\left\lceil \frac{28}{2 \times 5} \right\rceil - 1$. For Duty 1 and Duty 3, we set the initial values of C_{jkl} to $\left\lceil \frac{15}{2 \times 5} \right\rceil + 1$ and $\left\lceil \frac{22}{2 \times 5} \right\rceil + 1$,

respectively and varying the values according to the description given above. For instance, for Duty 1, by varying the values for C_{jkl} , there are 7 different problem instances generated.

In the following, we report a suite of computational results and analysis obtained from our mathematical models described above. Our mathematical models were implemented using ILOG OPL Studio 5.5 and executed on a Intel (R) Core (TM)² Duo CPU 2.33GHz with 1.96GB RAM that runs Microsoft Windows XP.

5.2 Results from Model I and IIa

First, results obtained from **Model I** for Random 1 and Random 2 problems are shown in Table 3. It is interesting to observe a two-point phase transition in the minimum number of unscheduled duties (column 2) with changing values of C_{jkl} . It remains unchanged over a sufficiently large range of values. As the value of C_{jkl} of the duty with the highest requirement

tends to $\left\lceil \frac{R_l \times \sum_{i \in I} A_{il}}{|K| \times |J|} \right\rceil$, the number of unscheduled duties starts to increase. For example, as

we decrease the C_{jkl} 's value for Duty 2 from 6 to 4 for Random 1 instances, the number of unscheduled duty remains zero, but when this value reaches 3, the number of unscheduled duties increases to 4. Then, when the number of resources available is set to

$\left\lceil \frac{R_l \times \sum_{i \in I} A_{il}}{|K| \times |J|} \right\rceil - 1$ for each activity $l \in L^C$, the number of unscheduled duties increases

drastically from 5 to 10 unscheduled duties. The same behavior was also observed for the other problem sets listed in Table 1.

From Table 3 again, we observe that the number of unscheduled duties is very low with respect to the number of scheduled duties. The percentage of unscheduled duties is on average less than 3% for Random 1 and Random 2 instances. Similar observations are made for other random problem sets. These results demonstrate the effectiveness of the optimization model on random hard instances. It is interesting to see these results in the light of the real-world case study problem instance where the percentage of unscheduled duties obtained is around 4.7%. This gives evidence to the hospital management that their resource capacity has reached a critical threshold as typified by problem instances 1g and 2i, and consequently towards a drastic reduction of performance if resources cannot come up to par with physician duties.

Table 3 also presents results on the extent of satisfiability of ideal schedule by the actual schedule, obtained from running **Model IIa**. These results are summarized in Figure 2, which visualizes the effect. Note that the left and right y-axes are on different scales, and measures the percentage of ideal, non-ideal scheduled and unscheduled; and the resource availability/requirement gapⁱ %Gap respectively. From the figure, we can infer that in order to ensure zero unscheduled duties (which is often a hard constraint, since doctor duties should not be unfulfilled), the total number of available resources for each activity $l \in L^C$ must be above the threshold $\left\lceil \frac{R_l \times \sum_{i \in I} A_{il}}{|K| \times |J|} \right\rceil + 1$ (see instances 1a, 1b and 1c). We can also infer that when the

ⁱ The resource availability/requirement gap is defined as

$$\%Gap = \frac{|K| \times |L| \times \sum_{j \in J} \sum_{k \in K} \sum_{l \in L^C} C_{jkl} - \sum_{i \in I} \sum_{l \in L^C} R_l \times A_{il}}{\sum_{i \in I} \sum_{l \in L^C} R_l \times A_{il}} \times 100\%.$$

It represents resource buffer, i.e. the proportion of total resource availability that exceeds the sum of resource requirement requested.

%Gap is decreased below 23%, the percentage of unscheduled duties will be doubled (see instances 1f vs 1g, also 2h vs 2i). Similar observations have been made for the rest of the problem instances. From the hospital administration standpoint, the latter result shows the critical resource availability threshold below which the degradation of service performance will be keenly felt.

Table 3. Summary of results for Models I and IIa

Problem Instances	Number of unscheduled duties	Number of scheduled duties		Percentage of unscheduled duties (%)	Percentage of scheduled duties (%)	
		Ideal	Non-ideal		Ideal	Non-ideal
Case study	7	139	4	4.7	92.6	2.7
Random 1a	0	196	4	0	98	2
Random 1b	0	192	8	0	96	4
Random 1c	0	192	8	0	96	4
Random 1d	4	186	10	2	93	5
Random 1e	5	179	16	2.5	89.5	8
Random 1f	5	178	17	2.5	89	8.5
Random 1g	10	173	17	5	86.5	8.5
Random 2a	0	196	4	0	98	2
Random 2b	0	196	4	0	98	2
Random 2c	0	196	4	0	98	2
Random 2d	0	194	6	0	97	3
Random 2e	0	194	6	0	97	3
Random 2f	3	184	13	1.5	92	6.5
Random 2g	3	184	13	1.5	92	6.5
Random 2h	3	183	14	1.5	91.5	7
Random 2i	10	176	14	5	88	7

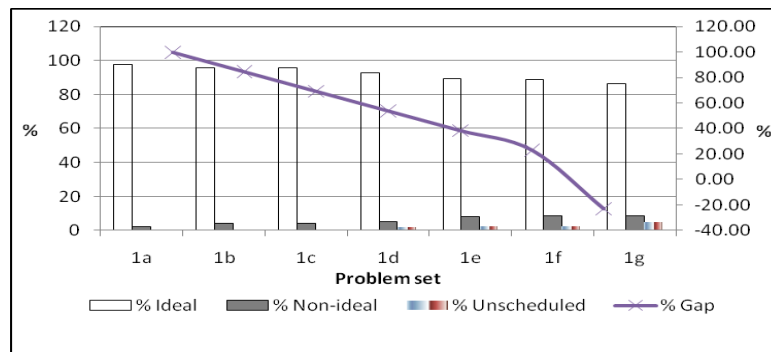


Figure 2. Parameter analysis of Random 1 Problem Set

Next, the result of the case study problem instance is compared with that of the actual allocation generated manually by the hospital, as summarized in Table 4. Although the number of unscheduled duties via manual allocation is better than the results obtained by Model IIa, the number of non-ideal scheduled duties is significantly higher than that of the model solution. We also found that two physicians have to cancel their days-off or shifts-off for other duties. This outcome is very undesirable since they might have external commitments that cannot be delayed or cancelled.

Table 4. Comparison between the manual allocation and model solution on a real case

	Manual allocation	Optimal solution
Number of unscheduled duties	5	7
Number of non-ideal scheduled duties	10	4
Number of physicians assigned duties during days-off or shifts-off	2	0

5.3 Results from Model IIb

Table 5 presents results obtained by running Model IIb against our problem instances. We observe that only Random 6 instances (where the percentage of *heavy* duties reaches 70%)

could not be solved to optimality within the computation time limit imposed. As such, we only report the best known solutions that could be obtained for these problem instances.

Table 5. Computational results of Model IIb

Problem instance	Number of unscheduled duties	Number of scheduled duties
Case study	8	142
Random 1a	0	200
Random 1b	0	200
Random 1c	0	200
Random 1d	4	196
Random 1e	5	195
Random 1f	5	195
Random 1g	10	190
Random 2a	0	200
Random 2b	0	200
Random 2c	0	200
Random 2d	0	200
Random 2e	0	200
Random 2f	3	197
Random 2g	3	197
Random 2h	3	197
Random 2i	10	190
Random 3a	0	200
Random 3b	0	200
Random 3c	0	200
Random 3d	0	200
Random 3e	0	200
Random 3f	5	195
Random 3g	9	191
Random 3h	9	191
Random 3i	10	190
Random 4a	0	200
Random 4b	0	200
Random 4c	0	200
Random 4d	0	200
Random 4e	0	200
Random 4f	0	200
Random 4g	0	200
Random 4h	3	197
Random 4i	6	194
Random 4j	6	194
Random 4k	10	190
Random 5a	0	200
Random 5b	0	200
Random 5c	0	200
Random 5d	0	200
Random 5e	0	200
Random 5f	0	200
Random 5g	0	200
Random 5h	0	200
Random 5i	0	200
Random 5j	3	197
Random 5k	8	192
Random 5l	8	192
Random 5m	10	190
Random 6a*	2	198
Random 6b*	2	198
Random 6c*	3	197
Random 6d*	3	197
Random 6e*	3	197
Random 6f*	3	197
Random 6g*	3	197
Random 6h*	3	197
Random 6i*	4	196
Random 6j*	4	196
Random 6k*	5	195
Random 6l*	6	194
Random 6m*	7	193

Problem instance	Number of unscheduled duties	Number of scheduled duties
Random 6n*	7	193
Random 6o*	23	177

*CPU time = 6 hours

Finally, using Model IIb, we perform sensitivity analysis of ergonomic constraints on schedule quality, which is of interest to the hospital administrator. These experiments are conducted over the same set of random instances as shown in Table 1 (see last column), and the findings are presented in Figure 3. The x-axis represents the percentage of heavy duties.

- The **square-dotted curve** shows the minimum total resource that should be available in order that the number of unscheduled duties will remain zero, while the **diamond-dotted curve** shows the corresponding resource requirement given by the input (refer to left y-axis). Comparing these curves we observe that across the board, a resource buffer of 20% of the total resource requirement should be set aside, in order to satisfy all ergonomic constraints. In other words, the cost of enforcing ergonomics constraints is 20% of resource requirement.
- Another observation is the impact on the number of unscheduled duties under fixed resource constraints. The **triangular-dotted curve** shows the number of unscheduled duties for problem instance 1c (refer to right y-axis). Note that as we increase the percentage of *heavy* duties, the total number of unscheduled duties increases gradually, until a certain threshold of 40%, when it is observed to increase sharply. This phase transition phenomenon is also observed in other random instances. It provides insights to the hospital administrator in terms of planning the limits of *heavy* duties for the physicians.

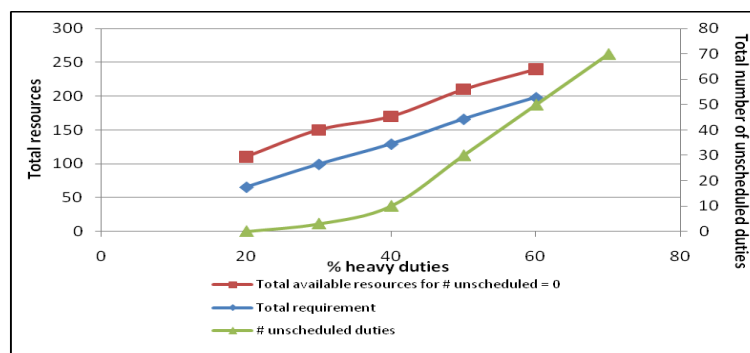


Figure 3. Sensitivity analysis on percentage of *heavy* duties

6 Conclusion

In this paper, we study the master physician scheduling problem experimentally. To the best of our knowledge, this is the first attempt that looks holistically at an entire range of physician duties quantitatively that enables hospital administrations to cope with increasingly stringent patient demands. Since the particular problem studied is representative of the Surgery Department of a large local government hospital, we believe our model requires minor customizations for use in other hospitals with similar constraints and preference structures.

We see many possibilities of extending the work. Our approach in this paper is purely optimization-based, including the handling of physician preferences. It will be interesting to investigate how other preference-handling methods (such as CP-nets) can be incorporated to model complex physician preferences. Similarly, one might also consider fairness constraints commonly seen in hospitals (Gendreau et al., 2007). Algorithmically, it would be interesting to tackle large-scale problems (such as the Random 6 instances) that cannot be solved by exact

optimization models with meta-heuristic or evolutionary approaches, or investigate multi-objective approaches since the problem can be modelled as a multi-objective problem.

Acknowledgements We like to thank the Department of Surgery, Tan Tock Seng Hospital (Singapore) for providing valuable comments and test situations. Hoong Chuin Lau was supported by a research grant from The Logistics Institute Asia Pacific, National University of Singapore.

References

1. Aggarwal, S. (1982). A focused review of scheduling in services. *European Journal of Operational Research*, 9(2), 114-121.
2. Bard, J. F., & Purnomo, H. W. (2005). Preference scheduling for nurses using column generation. *European Journal of Operational Research*, 164, 510-534.
3. Beaulieu, H., Ferland, J. A., Gendron, B., & Philippe, M. (2000). A mathematical programming approach for scheduling physicians in the emergency room. *Health Care Management Science*, 3, 193-200.
4. Beliën, J. (2007). Exact and heuristic methodologies for scheduling in hospitals: problems, formulations and algorithms. *4OR*, 5, 157-160.
5. Beliën, J., & Demeulemeester, E. (2005). Integrating nurse and surgery scheduling. In Proceedings of the 2nd Multidisciplinary International Scheduling Conference: Theory and Applications 2005, New York, USA, 18-21 July 2005, 408-409.
6. Bourdais, S., Galinier, P., & Pesant, G. (2003). A constraint programming application to staff scheduling in health care. In Rossi, F. (eds.), Principles and Practice of Constraint Programming (CP'03, Selected papers). *Lecture Notes in Computer Science*, 2833, 153-167, Springer, Heidelberg.
7. Burke, E. K., & Riise, A. (2008). Surgery allocation and scheduling. In Proceedings of the 7th International Conference of Practice and Theory of Automated Timetabling 2008, Montreal, Canada, 18-22 August 2008.
8. Burke, E. K., de Causmaecker, P., Berghe, G. V., & van Landeghem H. (2004). The state of the art of nurse rostering. *Journal of Scheduling*, 7(6), 441-499.
9. Buzon, I., & Lapierre, S. D. (1999). A tabu search algorithm to schedule emergency room physicians. Technical report, Centre de Recherche sur les Transports, Montréal, Canada.
10. Carter, M. W., & Lapierre, S. D. (2001). Scheduling emergency room physicians. *Health Care Management Science*, 4, 347-360.
11. Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., & Sier, D. (2004). An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127, 21-144.
12. Gendreau, M., Ferland, J., Gendron, B., Hail, N., Jaumard, B., Lapierre, S., Pesant, G., & Soriano, P. (2007). Physician scheduling in emergency rooms. In Burke, E. K., & Rudová, H. (eds.), The Practice and Theory of Automated Timetabling VI (PATAT'06, Selected papers). *Lecture Notes in Computer Science*, 3867, 53-66, Springer, Heidelberg.
13. Ogulata, S. N., Koyuncu, M., & Karakas, E. (2008). Personnel and patient scheduling in the high demanded hospital services: a case study in the physiotherapy service. *Journal of Medical Systems*, 32(3), 221-228.
14. Petrovic, S., & Berghe, G.V. (2008). Comparison of algorithms for nurse rostering problems. In Proceedings of the 7th International Conference of Practice and Theory of Automated Timetabling 2008, Montreal, Canada, 18 – 22 August 2008.
15. Rousseau, L. M., Gendreau, M., & Pesant, G. (2002). A general approach to the physician rostering problems. *Annals of Operations Research*, 115, 193-205.
16. Testi, A., Tanfani, E., & Torre, G. (2007). A three-phase approach for operating theatre schedules. *Health Care Management Science*, 10, 163-172.
17. Vassilacopoulos, G. (1985). Allocating doctors to shifts in an accident and emergency department. *Journal of the Operational Research Society*, 36, 517-523.

A hybrid harmony search for university course timetabling

Mohammed Azmi Al-Betar • Ahamad Tajudin Khader

Abstract. Combinations of metaheuristics to solve highly constrained combinatorial optimisation problems have recently become a promising research area. Some combinations emerge from integrating evolutionary algorithms and local based algorithms to strike the right balance between global exploration of the search region and local exploitation of promising regions. This study presents hybridization between harmony search algorithms and hill climbing optimisers to tackle the university course timetabling problem. The solution is essentially a harmony search algorithm. The hill climbing optimiser is responsible for improving the new harmony vector, obtained from harmony search, with a certain probability. Furthermore, inspired by the practical swarm optimisation, we modified the memory consideration operator to let it improvise rather than always working in the same way. Our algorithm converges significantly faster than does the classical harmony search algorithm. We evaluate our hybrid algorithm with a standard benchmark introduced for UCTP. Our results show that our algorithm can find higher quality solutions than can previous work.

1 Introduction

Timetabling problems are complex combinatorial optimisation problems classified as NP-hard [1]. The general requirement for timetabling problems “is to assign a set of entities (such as tasks, public events, vehicles, or people) to a limited number of resources over time, in such a way as to meet a set of pre-defined schedule requirements”[2]. Such problems appear in many forms such as school, examination, or course timetabling [3, 4, 2, 5]; employee timetabling [6]; transport timetabling [7]; and nurse rostering [8].

This paper addresses the university course timetabling problem (UCTP), which includes assigning sets of events to particular rooms and time slots on a weekly basis according to constraints. There are two kinds of timetabling constraints: hard and soft [3]. A timetabling

Mohammed Azmi Al-Betar

School of Computer Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia

E-mail: mohbetar@cs.usm.my

Ahamad Tajudin Khader

School of Computer Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia

E-mail: tajudin@cs.usm.my

solution must satisfy hard constraints in order to be feasible, but need not necessarily satisfy all soft constraints. In other words, soft constraints may be violated, and the quality of a timetable solution is optimised by satisfying soft constraints. The most common soft constraints reflect the preferences of students and teachers.

The UCTP has attracted attention from researchers in artificial intelligence and operational research for quite a long time. Many approaches have been proposed to tackle this problem. Because timetabling is similar to graph colouring problems (graph vertices correspond to timetable events, and colours correspond to time periods) [9, 10], the earliest approaches depended on graph colouring heuristics (that is, largest degree, largest enrolment, largest weighted degree, or largest saturating degree); these heuristics build a timetable by assigning events to valid time periods and rooms one by one according to a particular order. Modern approaches often use these heuristics to find a feasible timetable [11, 12], and also use them as low-level heuristics in hyper heuristics [13, 14]. Furthermore, they serve as a fuzzy inference rule in fuzzy multiple ordering [12].

Researchers have successfully applied metaheuristics in both local search and population-based approaches to the UCTP. Local search approaches (iterated local search [15], tabu search [16], simulated annealing [17], and others) make sequences of local changes to the initial (or *current*) solution guided by an objective function until they reach a locally optimal solution. The key to the success of local approaches is the definition of neighbourhood structures, which make more of the search space reachable. Population-based approaches, also called evolutionary algorithms, have also been successfully applied to the UCTP. Examples of such approaches include genetic algorithms [18], ant colony optimization [15], and harmony search [19]. Population-based approaches typically use recombination and randomisation rules, which mix different components of solutions (or individuals) with the hope of finding a ‘good’ mixture in the new solution(s). The following articles survey previous metaheuristics for the UCTP [3, 4, 2, 20, 5].

Hybridization of metaheuristics is an efficient solution for the UCTP. In a recent comprehensive survey for timetabling problems, Qu *et al.* [21] suggest that “There are many research directions generated by considering the hybridization of meta-heuristic methods particularly between population-based methods and other approaches”. Hybridization between genetic algorithms and local algorithms or the so-called memetic algorithm is a successful hybrid approach that tackled university timetabling problems [22]. Recently, Abdullah *et al.* [11] have tackled the UCTP using the hybrid evolutionary algorithm. The system of Abdullah *et al.* combines variable neighbourhood algorithms and genetic algorithms, and it obtained the best known result for UCTP problem instances at the time of publishing.

The harmony search algorithm (HSA), developed by Geem *et al.* [23] is a new metaheuristic population-based algorithm that imitates musical performance. HSA combines the key components of population-based and local based algorithms in a simple optimisation model [23, 24].

The HSA has strong components (operators) that can efficiently and effectively explore the search space. It combines some components of population-based algorithms (that is, memory consideration corresponding to recombination and random consideration which corresponding to mutation), which are responsible for global improvement, and some components of local search algorithms (that is, pitch adjustment corresponding to neighbourhood structures), which are responsible for local improvement.

Our previous work adapted the HSA to the UCTP [19]. The pitch adjustment operator is designed to work similarly to neighbourhood structures. At each run, this operator makes a

limited number of local changes to the new improvised solution, called “new harmony”, based on the pitch adjustment rate (PAR) and harmony memory consideration rate (HMCR). These local changes move the new harmony solution to the neighbouring solution as a ‘random walk’ acceptance criterion without considering the objective function value. Therefore, the HSA cannot fine-tune the search space region of new harmony to find a local optimal solution at this region. In this study, we introduce a Hybrid Harmony Search Algorithm (HHSA) to be applied to the UCTP. The hill climbing optimizer (HCO) is incorporated as a new operator of the HSA to explore the search space region of new harmony thoroughly to fine-tune this region. In addition, we propose an idea stemming from practical swarm algorithms, modified by a memory consideration operator, to lower the selection pressure of this operator. Our method dramatically improves the convergence of HSA. In the HHSA, the essential application of the HSA, the HCO serves as a new operator, controlled by a parameter we call ‘Hill Climbing Rate (HCR)’. Furthermore, inspired by practical swarm optimisation, we modified the functionality of the memory consideration operator to increase its improvisational ability.

This paper is organized as follows: Section 2 discusses the university course timetabling problem, and Section 3 explains the HSA. Section 4 presents the HSA for the UCTP and our hybridization strategy. Section 5 discusses our experimental results and compares them to the previous literature. In the final section, we present conclusions and future directions for our algorithm.

2 University course timetabling problem

2.1 Problem Description

Napier University course timetabling in Edinburgh was considered as an application problem by a Metaheuristics Network¹ (MN). The MN described the context of this problem as assigning certain events to suitable rooms and timeslots according to hard and soft constraints; a solution must satisfy hard constraints, while it should minimise violations of soft constraints.

The UCTP benchmark we consider was constructed by Socha *et al.* [15] who proposed eleven data instances using Paechter's generator² to mimic the same Napier University course timetabling. These data instances are grouped into three classes: five small, five medium, and one large, and they include three hard constraints:

- H1. Students must not be double-booked for events.
- H2. Room sizes and features must be suitable for assigned events.
- H3. Rooms must not be double booked for events.

And three soft constraints:

- S1. A student shall not have a class in the last slot of the day.
- S2. A student shall not have more than two classes in a row.
- S3. A student shall not have a single class on one day.

The main objective of this context is to satisfy the hard constraints and to minimise the violation of soft constraints.

¹ Metaheuristics Network official website “<http://www.metaheuristics.net/>”. (04 Feb 2008).

² Ben Paechter is a Professor in the School of Computing at Napier University, UK and a member of Metaheuristics Network. His official home page is “<http://www.dcs.napier.ac.uk/~benp/>”. (04 Feb 2008).

2.2 Problem formulation

In order to illustrate the bridge between the UCTP and harmony search algorithm, it is useful to discuss UCTP mathematically to overview the set of parameters and decision variables. Indeed, the formulation, presented here, could not be solved by exact approaches since the number of decision variables would be large. In the following subsections, the sets, parameters and decision variables are expressed before the problem formulation is introduced.

2.2.1 Sets and Parameters descriptions

- A set $E = \{1 \dots \bar{e}\}$ of events, each of which contains certain students and needs certain features.
- A set $R = \{1 \dots \bar{r}\}$ of rooms, each of which has a seat capacity and its own features.
- A set $S = \{1 \dots \bar{s}\}$ of students, each of whom enrolls in some events.
- A set $F = \{1 \dots \bar{f}\}$ of features, such as overhead projectors or special whiteboards.
- A set $P = \{1 \dots \bar{p}\}$ of timeslots where $p=45$ (5 days with 9 periods on each day).
- A set $\tilde{D} = \{\tilde{D}^1 \dots \tilde{D}^5\}$ of days, where each day has nine periods.
- ordered subsets $\tilde{P}^d$ of P . corresponding to a period in a day d . where
 $\tilde{P}^1 = \{p_1, p_2, \dots, p_9\}, \tilde{P}^2 = \{p_{10}, p_{11}, \dots, p_{18}\} \dots etc.$
- An ordered subset $\tilde{L}^d = \{p_9, p_{18}, p_{27}, p_{36}, p_{45}\}$ that contains the last periods of each day.
 $\tilde{L}^d \in P, d \in \tilde{D}$.
- $\bar{e}, \bar{r}, \bar{s}, \bar{f}, \bar{p}$ are the number of events, rooms, students, features, and timeslots respectively.
- s_r^R = the size of room r . $r \in R$.
- s_e^E = the number of students enrolled in event e . $e \in E$.
- $w_{f,e} = 1$ if event e requires feature f , 0 otherwise. $e \in E$ and $f \in F$.
- $y_{f,r} = 1$ if room r contains feature f , 0 otherwise. $r \in R$ and $f \in F$.
- $t_{s,e} = 1$ if student s is enrolled in event e , 0 otherwise. $s \in S$ and $e \in E$.

2.2.2 Decision variables

- x are binary decision variables indexed by events, rooms and timeslots. Their value $x_{e,r,p}$ is 1 if and only if event e occurred in room r and time period p . $e \in E, r \in R$ and $p \in P$.

- C_s^{ldp} (Last period of day) are decision variables indexed by students; their value indicating the number of violations of soft constraint S1 by student s . $s \in S$.
- C_s^{2R} (More than two events in a row) are decision variables indexed by students; their value indicates the number of violations of soft constraint S2 by student s . $s \in S$.
- C_s^{1d} (Single class in a day) are decision variables indexed by students; their value indicate to the number of violations of soft constraint S3 by student s . $s \in S$.
- $z_{s,d}$ are binary decision variables indexed by student and day; their value indicates that the student s has a single class in a day d . $s \in S$ and $d \in \tilde{D}$.

2.2.3 Formulation

The objective function of the obtained solution can be described as follows:

$$\text{Minimize} \quad \sum_{s \in S} (C_s^{ldp} + C_s^{2R} + C_s^{1d}) \quad (1)$$

The objective function is described in Eq. (1), C_s^{ldp} , C_s^{2R} , and C_s^{1d} consecutively describe the violations of the soft constraints S1,S2,S3 made against the will of each student s . When each violation occurs in the solution, it will be penalized by 1.

The hard constraints are described by Eqs. (2-6); these constraints are mandatory for any feasible solution.

$$\forall e \in E \quad \sum_{r \in R} \sum_{p \in P} x_{e,r,p} = 1 \quad (2)$$

$$\forall p \in P. \forall s \in S \quad \sum_{e \in E} \sum_{r \in R} t_{s,e} x_{e,r,p} \leq 1 \quad (3)$$

$$\forall r \in R. \forall p \in P \quad \sum_{e \in E} s_e^E x_{e,r,p} \leq s_r^R \quad (4)$$

$$\forall f \in F. \forall r \in R. \forall p \in P \quad \sum_{e \in E} w_{f,e} y_{f,r} x_{e,r,p} = 1 \quad (5)$$

$$\forall r \in R. \forall p \in P \quad \sum_{e \in E} x_{e,r,p} \leq 1 \quad (6)$$

Eq. (2) describes the implicit constraint, not mentioned in Section 2.1, which meant that timetable solution must be complete and each event must be presented once. Eq. (3) describes H1, which means that no student may be double-booked for events. Eqs. (4, 5) describe H2, which reflects room allocation with suitable capacity and features for events. Eq. (6) reflects H3, which explains that no room may be double-booked for events.

$$\forall s \in S \quad C_s^{ldp} = \sum_{e \in E} \sum_{r \in R} \sum_{q \in \tilde{L}} t_{s,e} x_{e,r,q} \quad (7)$$

$$\forall s \in S \quad C_s^{2R} = \sum_{\substack{i,j,k \in E \\ i \neq j \neq k}} \sum_{r \in R} \sum_{\substack{p,q,m \in P \\ q=p+1 \\ m=q+1}} t_{s,i} t_{s,j} t_{s,k} x_{i,r,p} x_{j,r,q} x_{k,r,m} \quad (8)$$

$$\forall s \in S. \forall d \in \tilde{D} \quad z_{s,d} = \begin{cases} 1 & \sum_{e \in E} \sum_{r \in R} \sum_{i \in \tilde{P}^d} t_{s,e} x_{e,r,i} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

$$\forall s \in S \quad C_s^{1d} = \sum_{\substack{d \in \tilde{D} \\ q \in \tilde{D}}} z_{s,q} \quad (10)$$

The decision variables C_s^{ldp} , C_s^{2R} , C_s^{ld} reflect the violation of soft constraints S1, S2, and S3, consecutively. These soft constraints are formalised in Eq. (7), which describes the soft constraint S1, Eq. (8) describes the soft constraint S2, and Eq. (10) describes the soft constraint S3. Eq. (9) is necessary for describing S3, which penalises students who have only attended a single event in a day by 1, while Eq. (10) calculates all violations of any students for all days.

3 Harmony search algorithm

Current metaheuristics originate from natural phenomena and mimic real situations. Examples include physical annealing in simulated annealing, ant behaviour in ant systems, vertebrate immune systems in artificial immune systems, human neural system in artificial neural networks, human memory in tabu search, and evolution in genetic algorithms. The HSA is a new metaheuristic algorithm developed by Geem *et al.* [23] to imitate the natural phenomenon of musical performance, in which musicians jointly tune the pitches of their instruments to find a euphonious harmony. This intense and prolonged musical process leads them to the perfect (or Nirvana) state.

The HSA is a scalable stochastic search mechanism, simple in its concept, and has few parameters, needing no derivational information at the initial stage [24]. It can find a trade-off between global improvement of search space and local improvement of the new harmony. Furthermore, HSA works at the component level rather than the individual level, which considers components (or decision variables) of the stored solutions (or harmony memory vectors) as essential elements to generate a new solution called the new harmony. The HSA has been successfully applied to several optimisation problems [25].

Algorithm 1 describes the basic HSA, which has five steps, as follows [23]:

- Step 1. Initialize the HSA and optimisation problem parameters.
- Step 2. Initialize the Harmony Memory (HM).
- Step 3. Improvise a new harmony.
- Step 4. Update the harmony memory.
- Step 5. Check the stopping criterion.

These steps will be explained similarly to Lee and Geem [24] in the next five subsections:

3.1 Initialize the HSA and optimisation problem parameters

In step 1, the optimisation problem can be specified as follows:

$$\min \{f(\mathbf{x}) | \mathbf{x} \in \mathbf{X}\} \quad \text{Subject to} \quad g(\mathbf{x}) \leq 0 \text{ and } h(\mathbf{x}) = 0 \quad (11)$$

Where $f(\mathbf{x})$ is the objective function; $\mathbf{x}$ is the set of each decision variable x_i , that is, $i \in \{1 \dots N\}$, $\mathbf{X}$ is the possible range of values for each decision variables, that is, $\mathbf{X}_i = \{x_i(1), x_i(2), \dots, x_i(K)\}$. N is the number of decision variables, and K is the number of possible values of decision variables. $g(\mathbf{x})$ is inequality constraint function and $h(\mathbf{x})$ is equality constraint function.

Furthermore, the parameters of the HSA required to solve the optimisation problem are also specified in this step: Harmony Memory Consideration Rate (HMCR), Harmony Memory Size (HMS) (that is, equivalent to population size), Pitch Adjustment Rate (PAR), and Number of Improvisations (NI) (that is, the maximum number of generations). The HSA parameters will be explained in more detail in the next steps.

Algorithm 1 The framework of a basic HSA

```

1: Input : the data instance P of the optimisation problem and the parameters for
   HSA(HMCR,PAR,NI,HMS).
2: Initialize-HM  $\{x^1, \dots, x^{HMS}\}$ 
3: Recognize  $x^{worst} \in \{x^1, \dots, x^{HMS}\}$ .
4: while not termination criterion specified by NI do
5:    $x^{NEW} \leftarrow \emptyset$  // new harmony vector
6:   for  $j=1, \dots, N$  do // N is the number of decision variables.
7:     if  $U(0,1) \leq \text{HMCR}$  then
8:       begin
9:          $x_j^{NEW} \in \{x_j^1, \dots, x_j^{HMS}\}$  // memory consideration
10:        if  $U(0,1) \leq \text{PAR}$  then // pitch adjustment
11:           $x_j^{NEW} \in N(x_j^{NEW})$  //  $N(x_j^{NEW})$  is the neighbouring values of variable  $x_j^{NEW}$ 
12:        end
13:      else //random consideration
14:         $x_j^{NEW} \in X_j$ 
15:      end if
16:    end for
17:    if  $f(x^{NEW}) \leq f(x^{worst})$  then
18:       $x^{worst} = x^{new}$ 
19:    end if
20:  end while
21: output: the best solution obtained so far.

```

3.2 Initialize the harmony memory

The harmony memory (HM) is a vector of solutions with size HMS as shown in Eq. (12). In this step, those solutions are randomly constructed and decreasingly filled to HM based on the values of the objective function.

$$HM = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_N^1 \\ x_1^2 & x_2^2 & \dots & x_N^2 \\ \vdots & \vdots & \dots & \vdots \\ x_1^{HMS} & x_2^{HMS} & \dots & x_N^{HMS} \end{bmatrix} \begin{bmatrix} f(x^1) \\ f(x^2) \\ \vdots \\ f(x^{HMS}) \end{bmatrix} \quad (12)$$

3.3 Improvise a new harmony

In this step, the HSA generates (improvises) a new harmony vector, $x^{NEW} = (x_1^{NEW}, x_2^{NEW}, x_3^{NEW}, \dots, x_N^{NEW})$, based on three operators: (1) memory consideration, (2) random consideration, and (3) pitch adjustment.

In the memory consideration, the decision variables of the new harmony randomly inherited the historical value stored in the HM with probability HMCR. Restating, the value of decision variable (x_1^{NEW}) is chosen from $\{x_1^1, x_1^2, x_1^3, \dots, x_1^{HMS}\}$ that stored in HM, the next decision variable (x_2^{NEW}) is chosen from $\{x_2^1, x_2^2, x_2^3, \dots, x_2^{HMS}\}$, the other decision variables, $(x_3^{NEW}, x_4^{NEW}, x_5^{NEW}, \dots)$, are chosen consecutively in the same way with probability HMCR where

($0 \leq HMCR \leq 1$). Other values that are not chosen depending on memory considerations with probability ($1-HMCR$) are selected according to their possible range by random consideration as in Eq. (13).

$$x_i^{NEW} = \begin{cases} x_i^{NEW} \in \{x_i^1, x_i^2, x_i^3, \dots, x_i^{HMS}\} & \text{with probability } HMCR \\ x_i^{NEW} \in X_i & \text{with probability } 1 - HMCR \end{cases} \quad (13)$$

The HMCR parameter is the probability of selecting one value of the decision variable, x_i^{NEW} , based on historical values stored in the HM. For instance, if ($HMCR = 0.90$), that indicates that the probability of selecting the value of decision variable from historic value in the HM with the probability is 90%, and the value of decision variable is selected from its possible range with a probability of 10%.

All decision variables, $x^{NEW} = (x_1^{NEW}, x_2^{NEW}, x_3^{NEW}, \dots, x_N^{NEW})$, chosen by memory considerations are examined to be pitch-adjusted with the probability PAR . Pitch adjustment makes a sequence of local changes (pitch adjustments) on the new harmony with probability PAR where ($0 \leq PAR \leq 1$). The size of these local changes is selected as in Eq. (14).

The pitch adjusting decision for x_i^{NEW} is:

$$x_i^{NEW} = \begin{cases} \text{Yes} & \text{with probability } PAR \\ \text{No} & \text{with probability } 1 - PAR \end{cases} \quad (14)$$

The values of decision variables not obtained by memory consideration with probability of ($1-PAR$) are not changed. The decision variable, suppose it is $x_i^{NEW}(k)$, that is, the k th element (or value) in x^{NEW} , that examined to be examined for pitch adjusting, is chosen as a neighbouring value with probability ($PAR \times HMCR$). For example, if $HMCR=90\%$ and $PAR=20\%$, the probability of selecting the neighbouring value of any decision variable is 18%. If the pitch adjusting decision for x_i^{NEW} is Yes, the pitch-adjusted value of $x_i^{NEW}(k)$ is:

$$x_i^{NEW} = x_i^{NEW}(k + m) \text{ where } m \text{ is the neighbouring index, } m \in \{\dots, -2, -1, 0, 1, 2, \dots\} \quad (15)$$

The HMCR parameters help the HSA to find globally improved solutions, and the PAR parameters help it to find locally improved solutions [26].

3.4 Update the harmony memory

If the new harmony vector, $x^{NEW} = (x_1^{NEW}, x_2^{NEW}, x_3^{NEW}, \dots, x_N^{NEW})$, has a better objective function value than the worst harmony stored in the HM, then it is included in the HM and the worst harmony vector is excluded from the HM.

3.5 Check the stopping criterion

Steps 3 and 4, presented in subsections 3.3 and 3.4, are repeated until the stop criterion (maximum number of improvisations) is met. This is determined by the NI parameter.

4 A hybrid harmony search algorithm for the UCTP

This section describes our HHSA for UCTP. First, we briefly describe HSA as presented in our previous work on UCTP; then, we illustrate the combination of HSA, HCO, and PSO. Figure 1 gives pseudo-code for each step of HHSA

4.1 The HSA for UCTP

The HSA represents the UCTP as a decision variable x indexed by rooms and time slots, where $x_{r,p}$ says whether a room contains event e in room r and timeslot p or -1 if it is empty. $r \in R$ and $p \in P$; See Eq. (16). The value of parameters $\bar{e}$ indicates the number of rooms, and $\bar{p}$ is the number of time slots.

$$x = \begin{bmatrix} x_{0,0} & x_{0,1} & \dots & x_{0,\bar{p}} \\ x_{1,0} & x_{1,1} & \dots & x_{1,\bar{p}} \\ x_{2,0} & x_{2,1} & \dots & x_{2,\bar{p}} \\ \dots & \dots & \dots & \dots \\ x_{\bar{e},0} & x_{\bar{e},1} & \dots & x_{\bar{e},\bar{p}} \end{bmatrix} \quad (16)$$

Step 1 specifies the UCTP parameters and possible range for each decision variable (that is, *event*). The objective function, $f(x)$, which describes the violations of soft constraints S1, S2, and S3, is defined in this step. The parameters of HSA (HMCR, PAR, HMS, and NI) are also selected in the same step.

In step 2, the random feasible timetables, $\{x^1, x^2, \dots, x^{HMS}\}$, are generated according to the HMS. These timetables will be decreasingly stored to the HM according to the objective function. See Eq. (17). The feasibility for all HM members is maintained with a method that combines largest weighted degree (LWD), backtracking algorithms, and the MultiSwap algorithm (for more details, see the pseudo-code in step 2 of Figure 1).

$$HM = \begin{bmatrix} x_{0,0}^1 & x_{0,1}^1 & \dots & x_{0,\bar{p}}^1 \\ x_{1,0}^1 & x_{1,1}^1 & \dots & x_{1,\bar{p}}^1 \\ x_{2,0}^1 & x_{2,1}^1 & \dots & x_{2,\bar{p}}^1 \\ \dots & \dots & \dots & \dots \\ x_{\bar{e},0}^1 & x_{\bar{e},1}^1 & \dots & x_{\bar{e},\bar{p}}^1 \end{bmatrix} \sim \begin{bmatrix} x_{0,0}^{HMS} & x_{0,1}^{HMS} & \dots & x_{0,\bar{p}}^{HMS} \\ x_{1,0}^{HMS} & x_{1,1}^{HMS} & \dots & x_{1,\bar{p}}^{HMS} \\ x_{2,0}^{HMS} & x_{2,1}^{HMS} & \dots & x_{2,\bar{p}}^{HMS} \\ \dots & \dots & \dots & \dots \\ x_{\bar{e},0}^{HMS} & x_{\bar{e},1}^{HMS} & \dots & x_{\bar{e},\bar{p}}^{HMS} \end{bmatrix} \quad (17)$$

In step 3, the new harmony (see Eq. (18)) is improvised based on memory consideration, random consideration, and pitch adjustment operators; if the complete new harmony is not obtained, then the repair strategy has to take over.

$$x^{NEW} = \begin{bmatrix} x_{0,0}^{NEW} & x_{0,1}^{NEW} & \dots & x_{0,\bar{p}}^{NEW} \\ x_{1,0}^{NEW} & x_{1,1}^{NEW} & \dots & x_{1,\bar{p}}^{NEW} \\ x_{2,0}^{NEW} & x_{2,1}^{NEW} & \dots & x_{2,\bar{p}}^{NEW} \\ \dots & \dots & \dots & \dots \\ x_{\bar{e},0}^{NEW} & x_{\bar{e},1}^{NEW} & \dots & x_{\bar{e},\bar{p}}^{NEW} \end{bmatrix} \quad (18)$$

In memory consideration, the value of the decision variable $x_{0,0}^{NEW}$ should be randomly selected from $x_{0,0}^{NEW} \in \{x_{0,0}^1, x_{0,0}^2, \dots, x_{0,0}^{HMS}\}$, the values of other decision variables,

$(x_{0,1}^{NEW}, x_{0,2}^{NEW}, x_{0,3}^{NEW}, \dots, x_{1,0}^{NEW}, \dots, x_{2,0}^{NEW}, \dots, x_{r,p}^{NEW})$, according to the basic HSA, should be selected sequentially in the same way [23, 24, 27].

In the UCTP case, this sequential process is not useful because, in our case, the feasibility for new harmony should be maintained (that is, we worked in feasible search space regions). The main difficulty of the UCTP lies in the attempt to find valid locations for each event. Indeed, each event has a specific number of valid locations in the HM members to be scheduled in the new harmony. For this reason, the selection of the new event, $x_{i,j}^{NEW}$, to be scheduled in new harmony should be based on available locations for this event in the HM members.

For UCTP, we propose the *smallest position algorithm* to select the appreciate event, $x_{i,j}^{New}$, to be scheduled in a new harmony with probability HMCR. In this algorithm, the event $e \in E$ is ordered iteratively based on the minimum valid locations available in the HM members. Let set $Q_e = \{Q_e^1, Q_e^2, \dots, Q_e^K\}$ be the HM members that have a valid location for event e to be scheduled in new harmony. The event e with the minimal number of valid places will be scheduled first. The valid location for event e will be randomly chosen from Q such that $x_e^{New} \in \{x_e^1, x_e^2, \dots, x_e^K\}$ with probability HMCR, where $(x_e^1 \in Q^1, x_e^2 \in Q^2, \dots, x_e^K \in Q^K)$ and K is the total number of HM members that have a valid location for event e .

In random consideration, the event e selected according to the smallest position algorithm is scheduled to the new harmony based on the possible available locations in the new harmony rather than in HM members with probability $(1-HMCR)$.

Indeed, in certain situations the two above operators cannot find a feasible timetable (this happens in some medium and large data instances). In this case, the algorithm initiates a repair process through one-level backtracking.

In the pitch adjustment operator, every decision variable (event), $x_{i,j}^{new}$, obtained by the memory consideration is examined to determine whether it should be pitch-adjusted with probability PAR. Moreover, for the UTCP, we design two pitch adjusting procedures, each one of which reflects a neighbourhood structure. Each pitch adjusting procedure is controlled by a particular PAR value as follows:

- 1- **PAR1-** PAR1 denotes the rate of local changes in the new harmony by *moving* an examined event e to a different location while preserving feasibility. More formally, let an event e be assigned to feasible timetable x^{NEW} in location k , $x_e^{NEW} = k$, based on memory consideration. Let location k' be randomly selected. Location k' is free and suitable for event e . The event e is moved to another location k' such that $(x_e^{NEW} = k')$.
- 2- **PAR2-** It denotes the rate of local changes in the new harmony by *swapping* the location of the examined event e and another event e' while preserving feasibility. More formally, let an examined event e be assigned to feasible timetable x^{NEW} in the location k , $x_e^{NEW} = k$, based on memory consideration. Let an event e' assigned to location k' , $x_{e'}^{NEW} = k'$, be randomly selected. Note that both locations k and k' are suitable for both events e and e' . The locations of event e and event e' are swapped such that $x_e^{NEW} = k'$ and $x_{e'}^{NEW} = k$.

These local changes of new harmony do a *random walk* without adopting any acceptance criteria like *first improvement* or *best improvement*. As such, this operator emphasises exploration rather than exploitation, which should be set to a small value.

Step 4 calculates the objective function for new harmony; if the objective function value of new harmony is better than the objective function of worst harmony in HM, then the new harmony is included in HM and the worst solution is excluded from the HM.

In *step 5*, steps 3 and 4 are repeated until the stop criterion determined by NI is satisfied.

4.2 Hybridization

This section discusses the hybridisation strategy between HSA and hill climbing optimiser (HCO), called the hybrid harmony search algorithm (HHSA). We then explain the need to hybridise the HHSA with Practical Swarm Optimisation (PSO) concepts to increase the convergence rate. Section 4.2.1 discusses the HHSA and section 4.2.2 discusses hybridisation between HHSA and PSO.

4.2.1 Hybridization with HCO

The HSA has three key components that generate a new harmony in every run. Memory and random considerations are the two components responsible for global improvement: memory consideration is a source of exploitation, and random considerations are a source of exploration [28]. The pitch adjustment component is the third component of HSA, which is responsible for local improvement. The size of the local improvements this component makes is determined mainly by the number of decision variables of each problem. The size is specified by the probability of HMCR and PAR: each decision variable that is selected according to memory consideration with probability HMCR is examined for potential pitch adjustment with probability PAR. Thus, the probability that each variable will be pitch-adjusted is $HMCR \times PAR$. As such, the amount of local improvement in the new harmony does not guarantee that the new harmony converges to the local optimal solution in each run. Furthermore, these local improvements are done randomly, which means that not all local changes might be responsible for the improvement in the new harmony.

Recently, Fesanghary *et al.* [29] have proposed a new hybridisation strategy for the harmony search algorithm to solve engineering optimization problems. The main motivations of this hybrid algorithm are to improve the solution quality and lower the computational cost of HSA. Their work incorporates sequential quadratic programming into the step 4 of the HSA to improve the quality of the new harmony vector and other vectors in HM. This incorporation process, as Fesanghary *et al.* explain, needs careful attention when the HMS is chosen for the problem, especially considering that the local optimiser is computationally expensive. Finally, Fesanghary *et al.* suggest that the applicability of hybrid harmony search can be found by using a few harmony memory vectors.

In fact, the HHSA presented in this study is similar to the memetic algorithm (MA), which is used mostly for timetabling problems by incorporating hill-climbing into genetic algorithm (GA) operators [22, 30]. One possible way to incorporate hill-climbing is 'Elaborate Encodings and Operators' [22], in which GA and the local search operator work in a feasible search space region. The main restriction to this strategy is that the exploratory ability of MA may be decreased.

In this study, we combine the hill climbing optimiser (HCO) with the HSA as a new operator (see Figure 1, step 3) which is controlled by a new parameter called 'hill climbing rate (HCR)'. The HCR is the rate of using the HCO to improve the new harmony vector relative to the number of improvisations (NI). The HCO starts with a feasible new harmony generated by the original HSA operators. In each run, the HCO explores the neighbourhood structures of the new harmony and moves to another neighbouring solution that has a better or equivalent objective function value. Indeed, we use the *best improvement* and *side walk* acceptance criteria to guide the HCO, which means that the HCO accepts only the best local change (the change with the best objective function value) among all possible local changes. *Side*

walk means the current solution walk to its neighbouring solution without changing the objective function. We consider five different neighbourhood structures, as follows:

- N1- Move one event from a timeslot to another empty one;
- N2- Swap two events in two different timeslots;
- N3- Exchange three events in three separate timeslots;
- N4- Move one event from a room to another empty one in the same timeslot;
- N5- Swap two events in two different rooms in the same timeslot.

The first three neighbourhood structures directly affect the solution cost, because they are directly linked to time slots and events. This means that the local changes that are linked to the interaction between time slots and events affect the soft constraints S1, S2, and S3, especially by penalising the events with relation to the time slot. The other two remaining neighbourhood structures do not affect the solution cost since they execute the same time slot process which is mainly useful in the *side walk*. These two neighbourhood structures very usefully reveal more options to the HCO in the coming iteration, because events may acquire a room with a better size and features.

4.2.2 Hybridization with PSO

Memory consideration is the most vital operator in HSA. Most of the decision variables in the new harmony are selected from the other vectors stored in HM. In the timetabling case, the decision variables (or events) are, by nature, highly correlated, which means that the random selection of the value of the event from any HM vector might influence other events in the new harmony, and this in turn might violate some soft constraints. Therefore, the HSA has a low probability of generating a good-quality new harmony when the value of each single event is different in the HM vectors, which may get stuck in a bottleneck. Actually, in HSA, this limitation has little or no effect on the convergence behaviour of HSA while, in the proposed HHSA, such a limitation does practically impede the effect of the speed of convergence. Here, the question below should be explicitly answered:

Why does memory consideration work well in HSA while it does not work well in HHSA?

To the best of our knowledge, the vectors stored in HM on HSA smoothly converge to the same search space region during the generation process, in which any combination (or mixture) done by memory consideration to generate a new harmony leads to the same search space region, and the functionality of this operator is thus efficient. Keep in mind that random consideration and pitch adjustment are the source of exploration to other search space regions. In contrast, the vectors stored in HM in the proposed HHSA belong to different search space regions that have a different structure than that used by HCO. Thus, the functionality of the memory consideration is lost in most cases where diversity is too high.

To address this limitation arising from the memory consideration of HHSA, we need another selection criterion that reduces the selection pressure of memory consideration. Therefore, the new harmony cost in most iterations can be improved.

In this study, we modify the memory consideration to use ideas from Particle Swarm Optimization (PSO)[31] as an auxiliary rule to select a promising value of the decision variables from the vectors stored in HM. This auxiliary concept mimics the best harmony among the HM vectors to construct the new harmony. Figure 1, step 3, presents the pseudo-code for the PSO concepts. The memory consideration selects the value(or location) of the decision variables primarily from the best harmony stored in harmony memory, if it is not feasible; the values of the decision variables are selected from any valid value stored in any HM vectors.

Similar to our work presented in this section, Omran and Mahdavi [32] proposed a new variation of HSA called ‘global best harmony search’. In their work, the pitch adjustment, which was modified to mimic the best so-far solution obtained in HM, is similar as the PSO concepts.

4.2.3 *Rationale for adopting the proposed hybrid mechanism.*

To better understand our motivation for introducing our hybrid algorithm, recall two terms from metaheuristic algorithms: exploration and exploitation. Any effective and robust metaheuristic algorithm must be based on two search techniques to find a global optimal solution: exploration to reach not-yet-visited regions in the search space when it is necessary, and exploitation to make use of the previous visited search space regions to yield a high quality solution. These terms contradict each other. The suitable balance between both exploration and exploitation must be achieved in any metaheuristics to find high quality solutions.

Any successful metaheuristic (local or population-based) has components responsible for exploring the search space efficiently and effectively and exploitation to fine-tune the regions of search space that were already visited. This is achieved by the objective function consideration. The more the metaheuristic component observes the objective function of previous state already visited during the search, the more the search drifts toward exploitation. In contrast, the more the same component observes randomness, the more the search is drawn to exploration [33]. Briefly, the pure hill climbing algorithm has an exploitation component with no exploration power [34].

The convergence rate of any metaheuristic relies on the balance between exploration and exploitation during the search. In general, local search metaheuristics are very effective in exploring the single search space region (exploitation) and finding the local optimal solution in this region (exploitation), yet they fail to explore the entire search space and may get stuck in a locally optimal solution (lack of exploration). On the other hand, the component power of population-based algorithms is very powerful in visiting multiple search space regions at the same time (exploration), yet they cannot fine-tune promising regions to find a globally optimal solution (exploitation) in each region. As pointed out earlier, recent surveys of metaheuristics direct the new researcher to turn to hybridization between population-based and local search algorithms. For example, Blum and Roli [33] write that “... In summary, population-based methods are better in identifying promising areas in the search space, whereas trajectory methods are better in exploring promising areas in the search space. Thus, metaheuristic hybrids that in some way manage to combine the advantage of population-based methods with the strength of trajectory methods are often very successful.”.

³ As an example for the metaheuristics component [33]: In a population based algorithm like a genetic algorithm or ant colony system, the ‘recombination operator’ works from the argument that a better offspring is obtained by a collection of good pieces from its parents. This means that the higher the recombination rate (that is, the crossover rate) is increased, the more the offspring is like its parent. As such, the recombination operator is useful in exploitation. The opposite case takes place if it decreases. Another example of a metaheuristic is the acceptance criterion in simulated annealing, which depends on the value of temperature and the objective function. The high temperature that means the chance of accepting the uphill move increases (the amount of exploration increases). The opposite happens if it decreases. A final example of metaheuristics is the tabu list in tabu search, in which the more the size of the tabu list (that is, tabu tenure) increases, the more the diversification power increases.

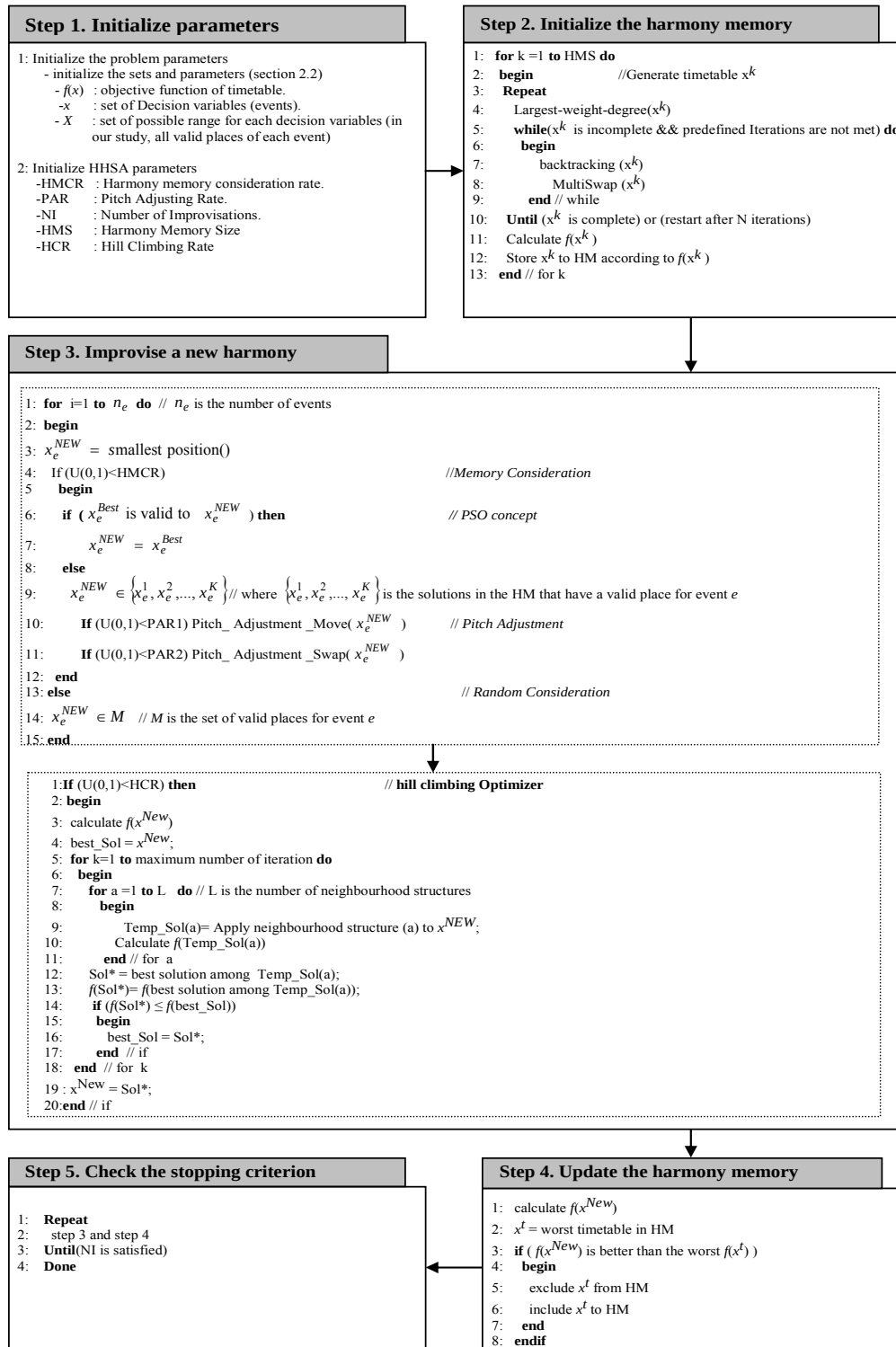


Figure 1. The steps and pseudo-code of our hybrid algorithm.

5 Experimental results and analysis

We now turn our attention to evaluate our hybrid methods. Section 5.1 discusses the problem instances, and Section 5.2 presents the comparison results of our hybrid method and other presented in the previous study that use the same problem instances. Section 5.3 analyses different individual and hybrid algorithms proposed in our study. Finally, we illustrate an empirical study of the performance of our hybrid method on a different HCR in Section 5.4.

We ran the experiments on an Intel 2 GHz Core 2 Quad processor with 2 GB of RAM. We implemented our method in Microsoft Visual C++ version 6.0 under Windows XP.

5.1 The problem instances

The UCTP data used in the experiments in this study are freely available at 'http://iridia.ulb.ac.be/~msampels/tt.data/', prepared by Socha *et al.*[15]. For the purposes of our study, we call them the 'Socha benchmark'. The 11 problem instances, which are grouped into five small problem instances, five medium problem instances and one large problem instance, have different levels of complexity and various sizes, as shown in Table 1. The solution to all problem instances must satisfy the hard constraints stated in Section 2.1. Furthermore, the solution cost is measured by the defined soft constraints violation.

Table 1
The characteristics of each class of Socha benchmarks [15].

Class	Small	Medium	Large
number of events	100	400	400
number of rooms	5	10	10
number of features	5	5	10
number of students	80	200	400
number of timeslots	45	45	45
approximate feature per room	3	3	5
percentage of feature use	70	80	90
maximum number of events per student	20	20	20
maximum number of students per event	20	50	100

5.1 Comparison with previous work

This section compares the best results obtained by HHSA to those applied by other methods in the literature to the same Socha benchmarks which are listed the Table 2 as follows:

- RRLS – Random Restart Local search [15].
- MMAS – MAX-MIN Ant System [15].
- THH – Tabu-search Hyper Heuristic [13].
- VNS – Variable Neighbourhood search [35].
- FMHO – Fuzzy Multiple Heuristic Ordering [12].
- GHH – Graph-based Hyper-Heuristic [14].
- RII – Randomised Iterative Improvement [36].
- HEA – Hybrid Evolutionary Approach [11].
- HSA – Harmony Search Algorithm [19].
- GD – Great Deluge [37].
- GDNLDLDR – Great Deluge with Non-Linear Decay Rate [37].
- HHSA – The proposed Hybrid Harmony Search Algorithm.

We compare the obtained results of HHSA with those of previous methods in Table 2. The best results reported among all methods are highlighted. The comparisons are meant to show the ability of HHSA to find high quality solutions to the UCTP. In short, the HHSA outperformed the previous methods in four out of five medium problem instances. The results of HHSA also shared the same best known results with RII, HEA, and some results introduced by MMSA, THH, VNS, and GDNLDR for small problem instances. In addition, HHSA reports the second-best result in the large problem instance.

Table 2

A comparison of results on the small / medium/large data instances with previous methods.

		Small 1	Small 2	Small 3	Small 4	Small 5	Medium 1	Medium 2	Medium 3	Medium 4	Medium 5	Large
RRLS	Avg.	8	11	8	7	5	199	202.5	-	177.5	-	-
MMAS	Avg.	1	3	1	1	0	195	184	284	164.5	219.5	851.5
THH	Best	1	2	0	1	0	146	173	267	169	303	1166
VNS	Best	0	0	0	0	0	317	313	375	247	292	-
FMHO	Best	10	9	7	17	7	243	325	249	285	132	1138
GHH	Best	6	7	3	3	4	372	419	359	348	171	1068
RII	Best	0	0	0	0	0	242	161	265	181	151	-
	Avg.	0	0	0	0	0	245	162.6	267.8	183.6	152.6	-
HEA	Best	0	0	0	0	0	221	147	246	165	130	529
	Avg.	0	0	0	0	0	224.4	150.6	252	167.8	135.4	552.4
HSA	Best	5	3	2	3	1	316	243	255	235	215	-
GD	Best	17	15	24	21	5	201	190	229	154	222	1066
GDNLDR	Best	3	4	6	6	0	140	130	189	112	141	876
HHSA	Best	0	0	0	0	0	105	102	135	122	123	671

We are particularly interested in comparing our results with HEA, which is the closest approach to our HHSA. The HEA hybridised variable neighbourhood structure with genetic algorithm (memetic algorithm), and our proposed algorithm also combined HCO with the harmony search algorithm. Both methods incorporate local search into population-based algorithms. Clearly, combining local search algorithms with population-based algorithms produced better results than others that were solely based either on local search or population-based algorithms separately in most cases. Note that any method can obtain good results by considering the right balance between global and local improvements incorporating both population and local search algorithms.

5.3 Performance of the proposed hybrid method

This section discusses the performance of five different methods proposed by us on the 11 Socha benchmarks. Specifically, these are the HHSA with PSO, HHSA without PSO, HSA with PSO, HSA without PSO, and HCO. The same program is used for all of these methods (in the second method and fourth one, the PSO codes are deactivated in the program), and the parameter configuration for each method is presented in Table 3. The first two methods are tested in the same parameter configurations to measure the effectiveness of the PSO on HHSA. Furthermore, the effectiveness of PSO upon the HSA is also studied by the experiments done in the third and fourth methods, which run in the same parameter configuration. Finally, we experiment with the HCO with more relaxed computational time than that used in HHSA.

Table 4 shows the comparison results obtained by the five proposed methods. The best results obtained are highlighted. Each method runs ten times for each Socha benchmark. The

best cost, the worst cost, and the approximate computational time among all runs are reported. We also report the average and standard deviation of the best results obtained for each of the ten runs. The results demonstrate that the HHSA with PSO can find higher quality solutions for all Socha benchmarks than those of others presented in this study.

Table 3

The parameters used for different methods

Parameters	HHSA with PSO	HHSA without PSO	HSA with PSO	HSA without PSO	HCO
NI	$1.5 \times e \times 3$	$1.5 \times e \times 3$	$e \times 5$	$e \times 5$	1
HMS	10	10	10	10	1
HMCR	98%	98%	98%	98%	100%
PAR1	2%	2%	2%	2%	0%
PAR2	4%	4%	4%	4%	0%
HCI*	$2 \times e \times 3$	$2 \times e \times 3$	0	0	$e \times 5$
HCR	30%	30%	0%	0%	100%

* HCI (the number of iteration of HCO).

Note that the numbers of iterations for both hybrid methods (HHSA with PSO and HHSA without PSO) are the same, but the results are significantly different, which demonstrates the effect of the PSO on HHSA convergence behaviour. See Figure 2 as an example of the effect of the PSO on HHSA, where the two trends are the average cost of HM vectors in each iteration. This means that the HHSA without PSO required more computational resources because it had to do additional iterations to be able to find the same results obtained by HHSA with PSO. For the large problem instance, the PSO helps the HHSA to find a feasible solution in each run.

Furthermore, we study the impact of PSO on the convergence behaviour of HSA. Here, we do ten runs for each Socha benchmark using two methods: the HSA with PSO and the HSA without PSO. Our results show that the PSO has not affected the ability of HSA to converge to a minimal solution, while it helps the HSA find a feasible solution for large problem instances.

Finally, we can observe that hybridisation between HSA as a global improvement search strategy and HCO as a local improvement search strategy, with auxiliary concepts from PSO, had a better solution cost than HSA without PSO and HCO, when each of them is individually running. This hybrid method stands out for its ability to strike a balance between global improvement and local improvement in a parallel optimisation environment.

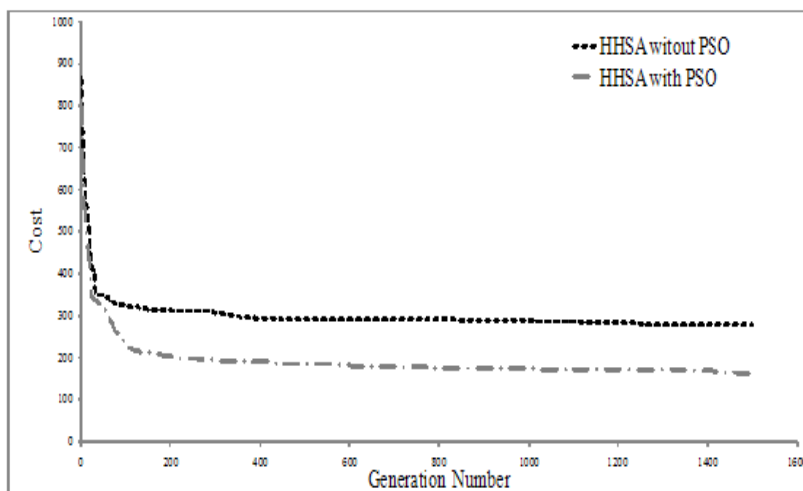


Figure 2. The impact of PSO on the HHSA for medium 1 problem instance.

Table 4

The performance of different variations of the proposed methods on each of the 11 Socha benchmarks.
(Note that h means hours and m means minutes).

		HHSA with PSO	HHSA without PSO	HSA with PSO	HSA without PSO	HCO
small1	Best	0	6	6	8	3
	Average	1.7	7.4	9.9	10.44	4.5
	Worst	3	9	12	13	6
	Std.dev.	1.159	0.96	2.02	2.06	1.08
	Time	18.14 m	17.04m	5.11m	4.83m	1.06m
Small2	Best	0	6	8	7	5
	Average	1.3	7.7	12.1	10.9	8.3
	Worst	3	9	16	13	12
	Std.dev.	1.33	0.94	2.18	1.79	2.40
	Time	32.45m	28.51m	4.55m	4.6m	12.70m
Small3	Best	0	4	5	7	5
	Average	2.2	4.4	9.2	10	7.4
	Worst	4	6	11	12	11
	Std.dev.	1.31	0.69	1.68	1.82	1.95
	Time	1.03 h	47.56m	20.7m	20.53m	3.20m
Small4	Best	0	4	4	2	3
	Average	2.77	5.1	7.1	5.8	6.1
	Worst	5	7	9	8	9
	Std.dev.	1.71	1.19	1.44	1.98	1.96
	Time	38.7 m	40.33m	12.19m	9.59m	41.65m
Small5	Best	0	0	3	0	2
	Average	0	0.4	5.4	3.7	4.8
	Worst	0	2	8	6	7
	Std.dev.	0	0.69	1.71	1.88	1.68
	Time	12.44 m	9.47m	25.10m	29.5m	7.17m
Medium1	Best	118	254	282	308	179
	Average	137.8	267.1	305.3	320.6	199.6
	Worst	155	283	321	336	226
	Std.dev.	12.56	8.96	11.17	9.93	14.22
	Time	2.06 h	1.56h	25.57m	22.27m	37.50m
Medium2	Best	106	270	336	282	177
	Average	126.9	275.7	351.7	300.1	200.77
	Worst	144	284	366	317	223
	Std.dev.	13.57	4.423	11.16	12.05	15.04
	Time	2.03 h	2.14h	21.4m	20.53m	32.55m
Medium3	Best	187	345	290	312	205
	Average	201.7	365.6	316.6	340.4	226.11
	Worst	224	379	328	352	243
	Std.dev.	12.15	9.21	11.87	13.47	11.85
	Time	1.54 h	1.35h	22.47m	24.23m	12.54m
Medium4	Best	130	255	245	251	163
	Average	145.2	271.44	266.4	271.9	196.6
	Worst	162	283	298	284	211
	Std.dev.	11.24	8.94	16.948	9.960	14.04
	Time	1.65 h	2.23 h	50.51m	34.73m	21.43m
Medium5	Best	155	199	240	266	277
	Average	180.3	227.5	273.2	292.7	296
	Worst	207	256	306	306	311
	Std.dev.	17.61	19.14	22.97	12.23	11.6619
	Time	4.6 h	4.5 h	35.93m	31.58m	31.9m
Large	Best	706	-	734	-	878
	Average	771.5	-	849	-	1015.8
	Worst	889	-	925	-	1168
	Std.dev.	64.22	-	58.83	-	96.5
	Time	2.36h	-	46 m	-	33.43m

5.4 Empirical study of the effect of HCR on convergence rate of HHSA

In this section, we aim to evaluate our hybrid method with PSO on different HCR settings, bearing in mind that the same parameter settings stated in Table 3 are used with different HCR settings. As we explained earlier, HCR is the probability of using the HCO in our hybrid method at each iteration, that is, NI. For example, if HCR=50%, this means that the HHSA uses the HCO in each iteration with probability of 50%. Values of HCR variations are settings of three scenarios: too small being selected 5%, medium selected 30% and too large selected 80%. These variations of HCR ran ten times for each Socha problem instance. The best cost, worst cost, and the computational time are recorded in Table 5. The average and standard deviation indicate the average and standard deviation of the best values among the HM vectors obtained from each single run. The best result between HCR variants is highlighted.

From the results recorded in Table 5, we can see that the best results of all Socha benchmarks are obtained from a high setting of HCR, which means that using HCO with a high HCR improves our algorithm's performance. The computational time is significantly different in the three variants, where increasing the HCR leads to increasing computation time with a significant improvement in the solution quality.

A further effect of the variant HCR on our hybrid method can be seen in Figure 3 which demonstrates statistical evidence that the HHSA is very effective in obtaining a high quality solution and that the HCR has a strong effect on the behaviour of HHSA. Box-plots in Figure 2 demonstrate the approximation distribution of cost values of the HCR variants for each Socha problem instance. The box shows the cost values between first and third quartiles, while the bold line inside the box denotes the median of obtained costs. The lower and upper cost values are indicated by whiskers. The asterisks denote highly extreme cost values.

6 Conclusions and future work

In this paper, we presented HHSA to tackle UCTP. We have incorporated HCO with harmony search algorithms as a new operator to improve the quality of new harmony in each run with probability HCR. This idea stems from an analogy with MA to find a trade-off between the locally improved solution from HCO and the globally improved solution from HSA. The results demonstrate that our algorithm can find a high-quality solution with a reasonable computational time, compared to previous work.

Having been inspired by PSO concepts, we modified the memory consideration operator to mimic the best harmony so far found that is stored in harmony memory. This modification influences the behaviour of the proposed HHSA, being able to configure a good-quality new harmony that might be further improved by HCO.

In short, our HHSA with PSO concepts could find the best results for four out of five medium problem instances prepared by Socha *et al.* It also shared the best results with some past literature research on small problem instances, while the second best result for large problem instances is obtained. This shows that hybridisation between local search algorithms and population-based algorithms is promising in this research area and can introduce high-quality solutions.

In fact, our previous work and the contribution in this paper work on a feasible region of search space. The limitation mainly implies the difficulty of producing a feasible solution in each iteration, which also affects running time. Our future aim is to study the hybrid harmony search algorithm over the entire search space (feasible and infeasible regions) to diversify searching even more efficiently.

As a recommendation, we propose that the hybrid harmony and harmony search algorithm to the UCTP problem has broken new ground and thus can be used with great efficiency in the optimisation process for combining global and local components in one algorithm, which can then be used in other scheduling problems like examination timetabling and school timetabling.

Table 5

The performance of the proposed methods using different HCR setting. (Note that h means hours and m means minutes).

		HHSA		
		HCR<=5%	HCR<=30%	HCR<=80%
small1	Best	0	0	0
	Average	2.1	1.7	1.4
	Worst	4	3	3
	Std.dev.	1.28	1.159	0.96
	Time	9.87 m	18.14 m	37.7m
Small2	Best	3	0	0
	Average	5.1	1.3	1.3
	Worst	7	3	3
	Std.dev.	1.72	1.33	1.15
	Time	9.66m	32.45m	53.4m
Small3	Best	1	0	0
	Average	3.4	2.2	1.7
	Worst	5	4	3
	Std.dev.	1.42	1.31	1.25
	Time	21.3m	1.03 h	3.2 h
Small4	Best	1	0	0
	Average	2.8	2.77	1.9
	Worst	5	5	4
	Std.dev.	1.61	1.71	1.52
	Time	15.3m	38.7 m	52.7m
Small5	Best	0	0	0
	Average	0.1	0	0
	Worst	1	0	0
	Std.dev.	0.31	0	0
	Time	7.3 m	12.44 m	13.66m
Medium1	Best	106	118	105
	Average	140.4	137.8	122.7
	Worst	156	155	133
	Std.dev.	14.36	12.56	8.95
	Time	1.4 h	2.06 h	7.277 h
Medium2	Best	121	106	102
	Average	139.3	126.9	119.9
	Worst	163	144	143
	Std.dev.	13.94	13.57	12.70
	Time	1.37 h	2.03 h	6.61 h
Medium3	Best	178	187	135
	Average	199.5	201.7	149.5
	Worst	216	224	166
	Std.dev.	11.04	12.15	9.87
	Time	1.4 h	2.54 h	6.61 h
Medium4	Best	124	130	122
	Average	139.2	145.2	135
	Worst	163	162	146
	Std.dev.	14.38	11.24	7.58
	Time	55.57m	1.65 h	3.81 h
Medium5	Best	163	155	123
	Average	194.3	180.3	165.2
	Worst	244	207	187
	Std.dev.	21.89	17.61	18.83
	Time	2.19 h	4.6 h	7.39 h
Large	Best	678	706	671
	Average	822.1	771.5	769.8
	Worst	963	889	869
	Std.dev.	87.85	64.22	70.4
	Time	1.62h	2.36h	4.54h

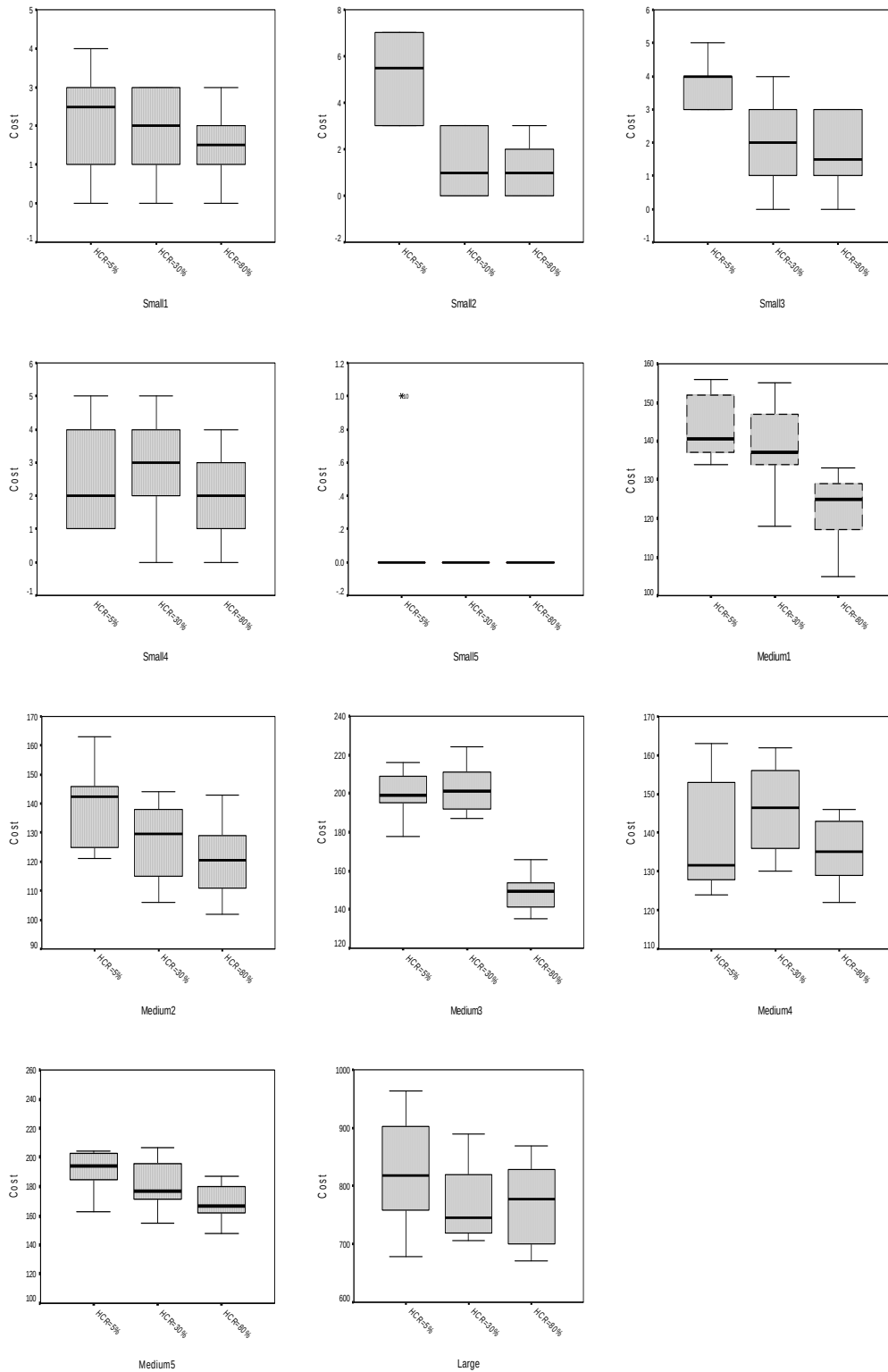


Figure 3. The result of the proposed hybrid method with variant HCR where HCR= 5%, HCR=30%, and HCR=80%.

References

1. M. Garey and D. Johnson. In: *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman, San Francisco, CA (1979).
2. R. Lewis, A survey of metaheuristic-based techniques for university timetabling problems, *OR Spectrum*, **30** (1), pp. 167–190 (2007).
3. E.K. Burke, J. Kingston, K. Jackson and R. Weare , Automated university timetabling: The state of the art, *Comp J* **40**(9), pp. 565–571(1997).
4. M.W. Carter, G. Laporte, Recent developments in practical course timetabling, in: E. K. Burke, M. Carter (Eds.), *Practice and Theory of Automated Timetabling II*, Lecture Notes in Computer Science, vol. **1408**, Springer, Berlin, pp. 3–19 (1997).
5. A. Schaerf, A survey of automated timetabling, *Artif Intelli Rev* **13**(2), pp. 87–127 (1999).
6. F. Glover and C. McMillan, The general employee scheduling problem: an integration of management science and artificial intelligence, *Computers Ops Res*, **13**(5), pp. 563–593 (1986).
7. A. Wren and J.M. Rousseau , Bus driver scheduling—an overview. In: J.R. Daduna, I. Branco and J.M.P. Paixao(eds), *Proceedings of Computer-Aided Transit Scheduling*, Lisbon, Portugal, Springer-Verlag, pp. 173–187 (1995).
8. B. Cheang, H. Li, A. Lim and B. Rodrigues, Nurse rostering problems—a bibliographic survey, *Eur J Opl Res* , **151**(3), pp. 447-460(2003).
9. D.C. Wood, A technique for colouring a graph applicable to large scale timetabling problems, *Comp J*, **12**(4), pp. 317-319 (1969).
10. D.J.A. Welsh and M. B. Powell, An upper bound for the chromatic number of a graph and its application to timetabling problems, *Comp J*, **10**(1), pp. 85-86 (1967).
11. S. Abdullah, E.K. Burke, B. McCollum , A hybrid evolutionary approach to the university course timetabling problem. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, Singapore, Sept., pp. 1764-1768 (2007).
12. H. Asmuni, E.K. Burke and J. Garibaldi, Fuzzy multiple heuristic ordering for course timetabling, In: *Proceedings of the 5th United Kingdom Workshop on Computational Intelligence*, pp. 302–309 (2005).
13. E.K. Burke, G. Kendall and E. Soubeiga, A tabu-search hyperheuristic for timetabling and rostering, *J Heuristics*, **9**(6), pp. 451-470 (2003).
14. E.K. Burke, B. McCollum, A. Meisels, S. Petrovic and R. Qu, A graph-based hyper-heuristic for educational timetabling problems, *Eur J Opl Res*, **176**(1), pp. 177-192 (2007).
15. K. Socha, K. Joshua and S. Michael, A max-min ant system for the university course timetabling problem. In *Proceedings of the 3rd International Workshop on Ant Algorithms (ANTS 2002)*. Lecture Notes in Computer Science, Vol. **2463**. Springer-Verlag, Berlin, pp. 1-13 (2002).
16. R. Alvarez-Valdes, E. Crespo and J. M. Tamarit, Design and implementation of a course scheduling system using tabu search, *Eur J Opl Res* **137**(3), pp. 512-523 (2002).
17. M. Chiarandini, M. Birattari, K. Socha and O. Rossi-Doria, An effective hybrid algorithm for university course timetabling, *J. of Scheduling*, **9**(5), pp.403-432 (2006).
18. R. Lewis and B. Paechter, New crossover operators for timetabling with evolutionary algorithms, In: Lofti A (ed) *The fifth international conference on recent advances in soft computing RASC2004*, Nottingham, England, pp. 189-194 (2004).
19. M.A. Al-Betar, A.T. Khader, T.A. Gani, A harmony search algorithm for university course timetabling. In: *7th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*, Montreal, Canada, August 18–22 (2008).
20. R. Lewis, Metaheuristics for university course timetabling, *Ph.D. Thesis*, August (2006).

21. R. Qu, E.K. Burke, B. McCullom, L. T. G. Merlot and S. Y. Lee, A survey of search methodologies and automated system development for examination timetabling, *J. of Scheduling*, **12**(1), pp. 55-89 (2009).
22. E.K. Burke and J. Landa Silva, The design of memetic algorithms for scheduling and timetabling problems. In: S. J., N. Krasnogor and W. Hart(Eds), *Recent Advances in Memetic Algorithms*, pp. 289–312 (2004).
23. Z.W. Geem, J.-H. Kim and G. Loganathan, A new heuristic optimization algorithm: Harmony search, *Simulation* **76**(2), pp. 60-68 (2001).
24. K.S. Lee and Z.W. Geem, A new structural optimization method based on the harmony search algorithm, *Comput Struct*, **82** (9-10), pp.781-798 (2004).
25. G. Ingram and T. Zhang, "Overview of applications and developments in the harmony search algorithm. *Music-inspired harmony search algorithm*, pp. 15-37(2009).
26. M.W. Carter, G. Laporte and S. Y. Lee, Examination timetabling: Algorithmic strategies and applications, *J Oper Res Soc*, **74**, pp. 373-3837 (1996).
27. M. Mahdavi, M. Fesanghary and E. Damangir, An improved harmony search algorithm for solving optimization problems, *Appl. Math. Comput.*, **188**(2), pp.1567-1579 (2007).
28. X.-S. Yang, Harmony search as a metaheuristic algorithm, *Music-inspired harmony search algorithm*, pp. 1-14 (2009).
29. M. Fesanghary, M. Mahdavi, M. Minary and Y. Alizadeh, Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems, *Comput. Methods Appl. Mech. Eng.*, **197**(33-40), pp. 3080–3091 (2008).
30. E.K. Burke, J. Newall and R. Weare, A memetic algorithm for university exam timetabling. In: E.K. Burke and P. Ross(eds), *Selected Papers from the 1st International Conference on the Practice and Theory of Automated Timetabling*, Lecture Notes in Computer Science Vol. **1153** Springer-Verlag, Berlin, pp. 241–250 (1996).
31. R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, pp. 39–43 (1995).
32. M.G.H. Omran and M. Mahdavi, Global-best harmony search, *Appl. Math. Comput.*, **198**(2), pp. 643-656 (2008).
33. C. Blum and A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, *ACM Comput. Surv.*, **35**(2), pp.268-308 (2003).
34. H.-G. Beyer, On the Explorative Power of ES/EP-like Algorithms, in V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben, (eds), *Evolutionary Programming VII Seventh Annual Conference on Evolutionary Programming (EP'98)*, Springer-Verlag, Berlin, Heidelberg, Germany, pp. 323-334 (1998).
35. S. Abdullah, E.K. Burke and B. McCollum, An investigation of variable neighbourhood search for university course timetabling. In *Proceedings of the 2nd Multidisciplinary Conference on Scheduling: Theory and Applications (MISTA)*, New York, USA, July, pp. 413-427 (2005).
36. S. Abdullah, E.K. Burke and B. McCollum, Using a randomised iterative improvement algorithm with composite neighbourhood structures for the university course timetabling problem. In: *Metaheuristics*, pp. 153-169 (2007).
37. D. Landa-Silva and J.H. Obit, Great deluge with non-linear decay rate for solving course timetabling problems, In: *Proceedings of the 4th International IEEE Conference on Intelligent Systems*, pp. 8-11-18-18 (2008).

A Guided Search Genetic Algorithm for the University Course Timetabling Problem

Sadaf Naseem Jat · Shengxiang Yang

Abstract The university course timetabling problem is a combinatorial optimisation problem in which a set of events has to be scheduled in time slots and located in suitable rooms. The design of course timetables for academic institutions is a very difficult task because it is an NP-hard problem. This paper proposes a genetic algorithm with a guided search strategy and a local search technique for the university course timetabling problem. The guided search strategy is used to create offspring into the population based on a data structure that stores information extracted from previous good individuals. The local search technique is used to improve the quality of individuals. The proposed genetic algorithm is tested on a set of benchmark problems in comparison with a set of state-of-the-art methods from the literature. The experimental results show that the proposed genetic algorithm is able to produce promising results for the university course timetabling problem.

1 Introduction

Timetabling is one of the common scheduling problems, which can be described as the allocating of resources for factors under predefined constraints so that it maximises the possibility of allocation or minimises the violation of constraints [14]. Timetabling problems are often complicated by the details of a particular timetabling task. A general algorithm approach to a problem may turn out to be incapable, because of certain special constraints required in a particular instance of that problem. In the university course timetabling problem (UCTP), events (subjects, courses) have to be set into a number of time slots and rooms while satisfying various constraints. Timetabling has become much more difficult to find the general and effective solution due to the diversity of the problem, the variance of constraints, and particular requirements from university to university according to the charac-

Sadaf Naseem Jat
Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, United Kingdom
E-mail: snj2@mcs.le.ac.uk

Shengxiang Yang
Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, United Kingdom
E-mail: s.yang@mcs.le.ac.uk

teristics. There is no known deterministic polynomial time algorithm for the UCTP. That is, the UCTP is an NP-hard combinatorial optimisation problem [12].

The research on timetabling problems has a long history. Over the last forty years, researchers have proposed various timetabling approaches by using constraint-based methods, population-based approaches (e.g., genetic algorithms (GAs), ant colony optimization, and memetic algorithms), meta-heuristic methods (e.g., tabu search, simulated annealing, and great deluge), variable neighbourhood search (VNS), and hybrid and hyper-heuristic approaches etc. A comprehensive review on timetabling can be found in [8,18] and recent research directions in timetabling are described in [5].

Several researchers have used GAs to solve course timetabling problems [20,21,1,23]. Rossi-Doria et al. [16] compared different meta-heuristics to solve the course timetabling problem. They concluded that conventional GAs do not give good results among a number of approaches developed for the UCTP. Hence, conventional GAs need to be enhanced to solve the UCTP. In this paper, a guided search genetic algorithm, denoted GSGA, is proposed for solving the UCTP, which consists of a guided search strategy and a local search technique. GAs rely on a population of candidate solutions [22]. If there is a good population, then chances increase to create a feasible and optimal solution. In GSGA, a guided search strategy is used to create offspring into the population based on an extra data structure. This data structure is constructed from the best individuals from the population and hence stores useful information that can be used to guide the generation of good offspring into the next population. In GSGA, a local search technique is also used to improve the quality of individuals through searching in three kinds of neighbourhood structures. In order to test the performance of the proposed GSGA, experiments are carried out on a set of benchmark problems in comparison with a set of state-of-the-art methods from the literature.

The rest of this paper is organised as follows. The next section briefly describes the UCTP. Section 3 presents the genetic algorithm proposed in this paper for the UCTP. Experimental results of comparing the proposed GA and other algorithms from the literature are reported and discussed in Section 4. Section 5 concludes this paper with discussions on the future work.

2 The University Course Timetabling Problem

According to Carter and Laporte [8], the UCTP is a multi-dimensional assignment problem, in which students and teachers (or faculty members) are assigned to courses, course sections or classes and events (individual meetings between students and teachers) are assigned to classrooms and time slots.

In a UCTP, we assign an event (courses, lectures) into a time slot and also assign a number of resources (students, rooms) in such a way that there is no conflict between the rooms, time slots and events. As mentioned by Rossi-Doria et al. [17], the UCTP problem consists of a set of n events (classes, subjects) $E = \{e_1, e_2, \dots, e_n\}$ to be scheduled in a set of 45 time slots $T = \{t_1, t_2, \dots, t_{45}\}$ (i.e., nine for each day in a five day week), a set of m available rooms $R = \{r_1, r_2, \dots, r_m\}$ in which events can take place, a set of k students $S = \{s_1, s_2, \dots, s_k\}$ who attend the events and a set of l available features $F = \{f_1, f_2, \dots, f_l\}$ that are satisfied by rooms and required by each event.

In addition, interrelationships between these sets are given by five matrices. The first matrix shows which event is attended by which students. The second matrix indicates whether two events can be scheduled in the same time slot or not. The third matrix gives the features

that each room possesses. The fourth matrix gives the features required by each event. The last matrix lists the possible rooms to which each event can be assigned.

Usually, a matrix is used for assigning each event to a room r_i and a time slot t_i . Each pair of (r_i, t_i) is assigned a particular number corresponding to an event. If a room r_i in a time slot t_i is free or no event is placed then “-1” is assigned to that pair. In this way we assure that there will be no more than one event assigned to the same pair so that one of the hard constraint will always been satisfied.

For the room assignment we use a matching algorithm described by Rossi-Doria [16]. For every time slot, there is a list of events taking place in it and a preprocessed list of possible rooms to which the placement of events can be occurred. The matching algorithm uses a deterministic network flow algorithm and gives the maximum cardinality matching between rooms and events.

In general, the solution to a UCTP can be represented in the form of an ordered list of pairs (r_i, t_i) , of which the index of each pair is the identification number of an event $e_i \in E$ ($i = 1, 2, \dots, n$). For example, the time slots and rooms are allocated to events in an ordered list of pairs like:

$$(2, 4), (3, 30), (1, 12), \dots, (2, 7),$$

where time slot 4 and room 2 are allocated to event 1, time slot 30 and room 3 are allocated to event 2, and so on.

The real world UCTP consists of different constraints: some are hard constraints and some are soft constraints. Hard constraints must not be violated under any circumstances, e.g. students cannot attend two classes at the same time. Soft constraints should preferably be satisfied, but can be accepted with a penalty associated to their violation, e.g. students should not attend more than two classes in a row. In this paper, we will test our proposed algorithm on the problem instances discussed in [16]. We deal with the following hard constraints:

- No student attends more than one events at the same time;
- The room is big enough for all the attending students and satisfies all the features required by the event;
- Only one event is in a room at any time slot.

There are also soft constraints which are penalised equally by their occurrences:

- A student has a class in the last time slot of a day;
- A student has more than two classes in a row;
- A student has a single class on a day.

The goal of the UCTP is to minimise the soft constraint violations of a feasible solution (a feasible solution means that no hard constraint violation exists in the solution). The objective function $f(s)$ for a timetable s is the weighted sum of the number of hard-constraint violations $\#hcv$ and soft-constraint violations $\#scv$, which was used in [17], as defined below:

$$f(s) := \#hcv(s) * C + \#scv(s) \quad (1)$$

where C is a constant, which is larger than the maximum possible number of soft-constraint violations.

3 The Guided Search Genetic Algorithm

GAs are a class of powerful general purpose optimisation tools that model the principles of natural evolution [11]. GAs have been used for timetabling since 1990 [10]. Since then,

Algorithm 1 The Guided Search Genetic Algorithm (GSGA)

```

1: input : A problem instance  $I$ 
2: set the generation counter  $g := 0$ 
   {initialize a random population}
3: for  $i := 1$  to population size do
4:    $s_i \leftarrow$  create a random solution
5:    $s_i \leftarrow$  solution  $s_i$  after applying LocalSearch()
6: end for
7: while the termination condition is not reached do
8:   if  $(g \bmod \tau) == 0$  then
9:     apply ConstructMEM() to construct the data structure MEM
10:   end if
11:    $s \leftarrow$  child solution generated by applying GuidedSearchByMEM() or the crossover operator
      with a probability  $\gamma$ 
12:    $s \leftarrow$  child solution after mutation with a probability  $P_m$ 
13:    $s \leftarrow$  child solution after applying LocalSearch()
14:   replace the worst member of the population by the child solution  $s$ 
15:    $g := g + 1$ 
16: end while
17: output : The best achieved solution  $s_{best}$  for the problem instance  $I$ 

```

there are a number of papers investigating and applying GA methods for the UCTP [8]. In this paper, we propose an optimization method based on GAs that incorporates a guided search strategy and a local search operator for the UCTP. The pseudocode of the proposed guided search GA for the UCTP is shown in Algorithm 1.

The basic framework of GSGA is a steady state GA, where only one child solution is generated per iteration/generation. In GSGA, we first initialize the population by randomly creating each individual via assigning a random time slot for each event according to a uniform distribution and applying the matching algorithm to allocate a room for the event. Then, a local search (LS) method as used in [9] is applied to each member of the initial population. The LS method uses three neighbourhood structures, which will be described in section 3.4, to move events to time slots and then uses the matching algorithm to allocate rooms to events and time slots. After the initialization of the population, a data structure (denoted *MEM* in this paper) is constructed, which stores a list of room and time slot pairs (r, t) for all the events with zero penalty (no hard and soft violation at this event) of selected individuals from the population. After that this *MEM* can be used to guide the generation of offspring for the following generations. The *MEM* data structure is re-constructed regularly, e.g., every τ generations.

In each generation of GSGA, one child is first generated either by using *MEM* or by applying the crossover operator, depending on a probability γ . After that, the child will be improved by a mutation operator followed by the LS method. Finally, the worst member in the population is replaced with the newly generated child individual. The iteration continues until one termination condition is reached, e.g., a preset time limit t_{max} is reached.

In the following sub-sections, we will describe in details the key components of GSGA respectively, including the *MEM* data structure and its construction, the guided search strategy, the mutation operator, and the local search method.

3.1 The *MEM* Data Structure

There have been a number of researches in the literature on using extra data structure or memory to store useful information in order to enhance the performance of GAs and other

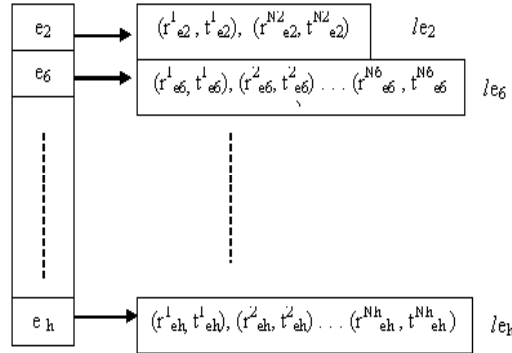


Fig. 1 Illustration of the data structure *MEM*.

Algorithm 2 *ConstructMEM()* – Constructing the data structure *MEM*

```

1: input : The whole population  $P$ 
2: sort the population  $P$  according to the fitness of individuals
3:  $Q \leftarrow$  select the best  $\alpha$  individuals in  $P$ 
4: for each individual  $I_j$  in  $Q$  do
5:   for each event  $e_i$  in  $I_j$  do
6:     calculate the penalty value of event  $e_i$  from  $I_j$ 
7:     if  $e_i$  is feasible (i.e.,  $e_i$  has zero penalty) then
8:       add the pair of room and time slot  $(r_{e_i}, t_{e_i})$  assigned to  $e_i$  into the list  $l_{e_i}$ 
9:     end if
10:   end for
11: end for
12: output : The data structure MEM

```

meta-heuristic methods for optimization and search [24,26,25]. In GSGA, we also use a data structure to guide the generation of offspring. Fig. 1 shows the details of the *MEM* data structure, which is a list of events and each event e_i has again a list l_{e_i} of room and time slot pairs. In Fig. 1, N_i represents the total number of pairs in the list l_{e_i} .

The *MEM* data structure is regularly reconstructed every τ generations. Algorithm 2 shows the outline of constructing *MEM*. When *MEM* is due to be reconstructed, we first select α best individuals from the population P to form a set Q . After that, for each individual $I_j \in Q$, each event is checked by its penalty value (Hard and soft constraints associated with this event). If an event has a zero penalty value, then we store the information corresponding to this event into *MEM*. For example, if the event e_2 of an individual $I_j \in Q$ is assigned room 2 at time slot 13 and has a zero penalty value, then we add the pair (2, 13) into the list l_{e_2} . Similarly, the events of the next individual $I_{j+1} \in Q$ are also checked by their penalty values. If the event e_2 in I_{j+1} has a zero penalty, then we add the pair of room and time slot assigned to e_2 in I_{j+1} into the existing list l_{e_2} . If for an event e_i , there is no list l_{e_i} existing yet, then the list l_{e_i} is added into the *MEM* data structure. Similar process is carried out for the selected Q individuals and finally the *MEM* data structure stores pairs of room and time slot corresponding to those events with zero penalty of the best individuals of the current population.

Algorithm 3 *GuidedSearchByMEM()* – Generating a child from *MEM*

```

1: input : The MEM data structure
2:  $E_s :=$  randomly select  $\beta * n$  events
3: for each event  $e_i$  in  $E_s$  do
4:   randomly select a pair of room and time slot from the list  $l_{e_i}$ 
5:   assign the selected pair to event  $e_i$  for the child
6: end for
7: for each remaining event  $e_i$  not in  $E_s$  do
8:   assign a random time slot and room to event  $e_i$ 
9: end for
10: output : A new child generated using the MEM data structure

```

This *MEM* data structure is then used to generate offspring for the next τ generations before re-constructed. We update *MEM* every τ generations instead of every generation in order to make a balance between the solution quality and the computational time cost of the proposed GSGA.

3.2 Generating a Child by the Guided Search Strategy

In GSGA, a child is created through the guided search by *MEM* or a crossover operator with a probability γ . That is, when a new child is to be generated, a random number $\rho \in [0.0, 1.0]$ is first generated. If ρ is less than γ , *GuidedSearchByMEM()* (as shown in Algorithm 3) will be used to generate the new child; otherwise, a crossover operation is used to generate the new child. Below we first describe the procedure of generating a child through the guided search by *MEM* and then describe the crossover operator.

If a child is to be created using the *MEM* data structure, we first select a set E_s of $\beta * n$ random events to be generated from *MEM*. After that, for each event e_i in E_s , we randomly select a pair of (r_{e_i}, t_{e_i}) from the list l_{e_i} that corresponds to the event e_i and assign the selected pair to e_i for the child. If there is an event e_i in E_s but there is no the list l_{e_i} in *MEM*, then we randomly assign a room and time slot from possible rooms and time slots to e_i for the child. This process is carried out for all the events in E_s . For those remaining events that are not present in E_s , they are assigned random rooms and time slots.

If a child is to be generated using the crossover operator, we first select two individuals from the population as the parents by the tournament selection of size 2. Then, we exchange the time slots between the two parents and allocate rooms to events in each non-empty time slot.

3.3 Mutation

After a child is generated by using either *MEM* or crossover, a mutation operator is used with a probability P_m . The mutation operator first randomly selects one from three neighbourhood structures N1, N2 and N3, which will be described in Section 3.4, and then make a move within the selected neighbourhood structure.

3.4 Local Search

After mutation, a local search (LS) method is applied on the child solution for possible improvement. Algorithm 4 summarises the LS scheme used in GSGA. LS works on all

Algorithm 4 *LocalSearch()* – Search the neighbourhood for improvement

```

1: input : Individual I from the population
2: for  $i := 1$  to  $n$  do
3:   if event  $e_i$  is infeasible then
4:     if there is untried move left then
5:       calculate the moves: first N1, then N2 if N1 fails, and finally N3 if N1 and N2 fail
6:       apply the matching algorithm to the time slots affected by the move and delta evaluate the result.
7:       if moves reduce hard constraints violation then
8:         make the moves and go to line 3
9:       end if
10:    end if
11:  end if
12: end for
13: if no any hard constraints remain then
14:   for  $i := 1$  to  $n$  do
15:    if event  $i$  has soft constraint violation then
16:      if there is untried move left then
17:        calculate the moves: first N1, then N2 if N1 fails, and finally N3 if N1 and N2 fail
18:        apply the matching algorithm to the time slots affected by the move and delta evaluate the
19:        result
20:        if moves reduce soft constraints violation then
21:          make the moves and go to line 14
22:        end if
23:      end if
24:    end if
25:  end for
26: output : A possibly improved individual I

```

events. Here, we suppose that each event is involved in soft and hard constraint violations. LS works in two steps and is based on three neighbourhood structures, denoted as N1, N2, and N3. They are described as follows:

- N1: the neighbourhood defined by an operator that moves one event from a time slot to a different one
- N2: the neighbourhood defined by an operator that swaps the time slots of two events
- N3: the neighbourhood defined by an operator that permutes three events in three distinct time slots in one of the two possible ways other than the existing permutation of the three events.

In the first step (line 2-12 in Algorithm 4), LS checks the hard constraint violations of each event while ignoring its soft constraint violations. If there are hard constraint violations for an event, LS tries to resolve them by applying moves in the neighbourhood structures N1, N2, and N3 orderly¹ until a termination condition is reached, e.g., an improvement is reached or the maximum number of steps s_{max} is reached, which is set to different values for different problem instances. After each move, we apply the matching algorithm to the time slots affected by the move and try to resolve the room allocation disturbance and delta-evaluate the result of the move (i.e., calculate the hard and soft constraint violations before and after the move). If there is no untried move left in the neighbourhood for an event,

¹ For the event being considered, potential moves are calculated in a strict order. First, we try to move the event to the next time slot, then the next, then the next, etc. If this search in N1 fails, we then search in N2 by trying to swap the event with the next one in the list, then the next one, and so on. If the search in N2 also fails, we try a move in N3 by using one different permutation formed by the event with the next two events, then with the next two, and so on.

Table 1 Three groups of problem instances

Class	Small	Medium	Large
Number of events	100	400	400
Number of rooms	5	10	10
Number of features	5	5	10
Per room approximate features	3	3	5
Percentage (%) of features used	70	80	90
Number of students	80	200	400
Maximum events per student	20	20	20
Maximum students per event	20	50	100

LS continues to the next event. After applying all neighbourhood moves on each event, if there is still any hard constraint violation, then LS will stop; otherwise, LS will perform the second step (lines 13-25 in Algorithm 4).

In the second step, after reaching a feasible solution, the LS method is used to deal with soft constraints. LS performs a similar process as in the first step on each event to reduce its soft constraint violations. For each event, LS tries to make moves in the neighbourhood N1, N2, and/or N3 orderly without violating the hard constraints. For each move, the matching algorithm is applied to allocate rooms to affected events and the result is delta-evaluated. When LS finishes, we get a possibly improved and feasible individual.

At the end of each generation, the obtained child solution replaces the worst member of the population to make a better population in the next generation.

4 Experimental Study

The program is coded in GNU C++ with version 4.1 and run on a 3.20 GHz PC. We use a set of benchmark problem instances to test our algorithm, which were proposed by Ben Paechter for the timetabling competition, see [15]. Although these problem instances lack many of the real world problem constraints and issues [13], they allow the comparison of our approach with current state-of-the-art techniques on them.

Table 1 represents the data of timetabling problem instances of three different groups: 5 small instances, 5 medium instances, and 1 large instance. The parameters for GSGA are set as follows: the population size *pop-size* is set to 50, $\alpha = 0.2 * pop-size = 10$, $\beta = 0.6$, $\gamma = 0.8$, $\tau = 20$, and $P_m = 0.5$. In the local search, the maximum number of steps per local search *s_{max}* is set to different values for different problem instances, which are 200 for small instances, 1000 for medium instances, and 2000 for the large instance respectively. There were 50 runs of the algorithm for each problem instance. For each run, the maximum run time *t_{max}* was set to 90 seconds for small instances, 900 seconds for medium instances, and 9000 seconds for the large instance.

We compare our GSGA with other algorithms on the 11 timetabling problem instances. The algorithms compared in the table are described as follows:

- GSGA: The guided search genetic algorithm proposed in this paper
- RIIA: The randomised iterative improvement method by Abdullah et al. [2]. They presented a composite neighbourhood structure with a randomised iterative improvement algorithm.
- VNS: The variable neighbourhood search by Abdullah et al. [3]. In [3], they used a variable neighbourhood search approach based on the random-descent local search with an exponential Monte Carlo acceptance criteria.

Table 2 Comparison of algorithms on small and medium problem instances

UCTP	GSGA		RIIA	GALS	GBHH	VNS	THHS	LS	EA	AA	FA
	Best	Med									
S1	0	0	0	2	6	0	1	8	0	1	10
S2	0	0	0	4	7	0	2	11	3	3	9
S3	0	0	0	2	3	0	0	8	0	1	7
S4	0	0	0	0	3	0	1	7	0	1	17
S5	0	0	0	4	4	0	0	5	0	0	7
M1	240	242.5	242	254	372	317	146	199	280	195	243
M2	160	164	161	258	419	313	173	202.5	188	184	325
M3	242	245	265	251	359	357	267	77.5%In	249	248	249
M4	158	161	181	321	348	247	169	177.5	247	164.5	285
M5	124	126.5	151	276	171	292	303	100%In	232	219.5	132
L	801	822	100%In	1027	1068	100%In	80%In	100%In	100%In	851.5	1138

- THHS: The tabu-based hyper-heuristic search by Burke et al. [6]. They introduced a tabu-search hyper heuristics where a set of low level heuristics compete with each other. This approach was tested on the course timetabling and nurse rostering problems.
- EA: The evolutionary algorithm (EA) by Rossi-Doria et al. [16]. They used a local search with the EA to solve the UCTP and also compared several metaheuristics methods on the UCTP.
- GALS: The GA with local search by Abdullah and Turabieh [1]. They tested a GA with a repair function and local search on the UCTP.
- LS: The local search method by Socha et al. [19]. They used a random restart local search for the UCTP and compared it with an ant algorithm.
- AA: The ant algorithm used by Socha et al. [19]. They developed a first ant colony optimization algorithm with the help of construction graph and a pheromone model appropriate for the UCTP.
- FA: The fuzzy algorithm by Asmuni et al. [4]. In [4], Asmuni et al. focused on the issue of ordering events by simultaneously considering three different heuristics using fuzzy methods.
- GBHH: The graph-based hyper heuristic by Burke et al. [7]. They employed tabu search with graph-based hyper-heuristics for the UCTP and examination timetabling problems.

Table 2 gives the comparison of the experimental results of our algorithm with the available results of other algorithms in the literature on the small and medium timetabling problem instances. In the table, *S1* represents small instance 1, *S2* represents small instance 2, and so on, and *M1* represents medium problem instance 1, *M2* represents medium problem instance 2, and so on, and *L* represents large instance. In Table 2, the term “%In” represents the percentage of runs that failed to obtain a feasible solution. “Best” and “Med” mean the best and median result among 50 runs respectively. We present the best result among all algorithms for each UCTP instance in the bold font.

From Table 2, it can be seen that our proposed GSGA is better than the fuzzy algorithm [4] and graph based approach [7] on all the 11 small, medium, and large problem instances. GSGA outperforms VNS [3], RIIA [2], and EA [16] on all the medium problem instances and ties them on some or all of the small problem instances. It also gives better results than local search [19] on 10 out of the 11 problem instances and is better than the ant algorithm [19] on 9 problem instances and ties it on S5. When comparing with the tabu-based hyper heuristic search [6], GSGA performs better or the same on all the problem instances except

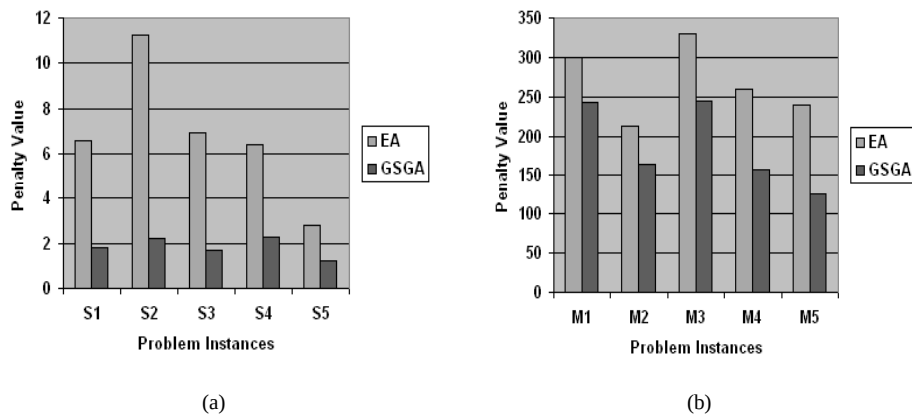


Fig. 2 Comparison of GSGA with the EA from [16] on (a) small and (b) medium problems.

Table 3 The *t*-test results of comparing GSGA against the EA by Rossi-Doria et al. [16]

function	S1	S2	S3	S4	S5
<i>t</i> -test	8.8874	8.0551	9.5634	7.7520	4.5521
function	M1	M2	M3	M4	M5
<i>t</i> -test	6.5912	5.0794	4.5667	17.6783	18.4526

on M1. Finally, the results of our approach is better then the GALS approach [1] on all medium and large instances and ties on S4.

We are interested to compare the results of GSGA with EA to show that the guided serach approach can help to minimise the penalty values and give better results for UCTP in comparison with conventional EAs employed in [16]. Fig. 2 shows the average penalty value over 50 runs of the EA by Rossi-Doria et al. [16] and our proposed GSGA on the small and medium UCTP instances, respectively. The results of the *t*-test statistical comparison of GSGA against their EA are shown in Table 3. The *t*-test statistical comparison is carried out with 98 degrees of freedom at a 0.05 level of significance. From Fig. 2 and Table 3, it can be seen that the performance of GSGA is significantly better than the EA by Rossi-Doria et al. on all small and medium problems. These results show that by integrating proper guided search techniques the performance of GAs for the UCTP can be greatly improved.

5 Conclusion and Future work

This paper presents a guided search genetic algorithm, i.e., GSGA, to solve the university course timetabling problem, where a guided search strategy and a local search technique are integrated into a steady state genetic algorithm. The guided search strategy uses a data structure to store useful information, i.e., a list of room and time slot pairs for each event that is extracted from the best individuals selected from the population and has a zero penalty value. This data structure is used to guide the generation of offspring into the next population. In GSGA, a local search technique is also used to improve the quality of individuals through searching three neighbourhood structures. To our knowledge this is the first such algorithm aimed at this problem domain.

In order to test the performance of GSGA, experiments are carried out based on a set of benchmark problems to compare GSGA with a set of state-of-the-art methods from the literature. The experimental results show that the proposed GSGA is competitive and work reasonably well across all problem instances in comparison with other approaches studied in the literature. With the help of the guided search strategy, GSGA is capable of finding (near) optimal solutions for the university course timetabling problem and hence can act as a powerful tool for the UCTP.

Future work includes further analysis of the contribution of individual components (local search and guided search) toward the performance of GSGA. Improvement of genetic operators and new neighbourhood techniques based on different problem constraints will also be investigated. We believe that the performance of GAs for the UCTP can be improved by applying advanced genetic operators and heuristics. The inter-relationship of these techniques and a proper placement of these techniques in a GA may lead to a better performance.

References

1. S. Abdullah and H. Turabieh. Generating university course timetable using genetic algorithm and local search. *Proc. of the 3rd Int. conf. on Hybrid Information Technology*, pp. 254-260, 2008.
2. S. Abdullah, E. K. Burke, and B. McCollum. Using a randomised iterative improvement algorithm with composite neighbourhood structures. *Proc. of the 6th Int. Conf. on Metaheuristic*, pp. 153-169, 2007.
3. S. Abdullah, E. K. Burke, and B. McCollum. An investigation of variable neighbourhood search for university course timetabling. *Proc. of the 2nd Multidisciplinary Conference on Scheduling: Theory and Applications*, pp. 413-427, 2005.
4. H. Asmuni, E. K. Burke, and J. M. Garibaldi. Fuzzy multiple heuristic ordering for course timetabling. *Proc. of the 5th UK Workshop on Comput. Intell.*, pp. 302-309, 2005.
5. E. K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operation Research*, **140**(2): 266-280, 2002.
6. E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics*, **9**(6): 451-470, 2003.
7. E. K. Burke, B. MacCloum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper heuristic for timetabling problems. *European Journal of Operation Research*, **176**: 177-192, 2006.
8. M. W. Carter and G. Laporte. Recent developments in practical course timetabling. *Proc. of the 2nd Int. Conf. on Practice and Theory of Automated Timetabling*, LNCS 1408, pp. 3-19, 1998.
9. M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, **9**(5): 403-432, 2006.
10. A. Colomi, M. Dorigo, and V. Maniezzo. Genetic algorithms - A new approach to the timetable problem. In Akgul et al. (eds.), *NATO ASI Series, Combinatorial Optimization*, Lecture Notes in Computer Science, F(82), pp. 235-239, 1990.
11. L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.
12. S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, **5**(4): 691-703, 1976.
13. B. McCollum. University Timetabling: Bridging the Gap between Research and Practice. *Proc of the 6th Int Conf on the Practice and Theory of Automated Timetabling*, pp. 15-35, 2006.
14. N. D Thanh. Solving timetabling problem using genetic and heuristics algorithms *Journal of Scheduling*, **9**(5): 403-432, 2006.
15. <http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.html>
16. O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Stützle. A comparison of the performance of different metaheuristics on the timetabling problem. *Lecture Notes in Computer Science 2740*, pp. 329-351, 2002.
17. O. Rossi-Doria and B. Paechter. A memetic algorithm for university course timetabling. *Proceedings of Combinatorial Optimization (CO 2004)*, pp. 56. 2004.
18. A. Scheerf. A survey of automated timetabling. *Artificial Intelligence Review*, **13**(2): 87-127, 1999.
19. K. Socha, J. Knowles, and M. Samples. A max-min ant system for the university course timetabling problem. *Proc. of the 3rd Int. Workshop on Ant Algorithms (ANTS 2002)*, LNCS 2463, pp. 1-13, 2002.

20. W. Erben, J. Keppler. A genetic algorithm solving a weekly course timetabling problem. *Proc. of the 1st Int. Conf. on Practice and Theory of Automated Timetabling*, LNCS 1153, pp. 198-211, 1995.
21. P. Pongcharoen, W. Promtiet, P. Yenradee, and C. Hicks. Stochastic Optimisation Timetabling Tool for University Course Scheduling. *International Journal of Production Economics*, **112**: 903-918, 2008.
22. K. Sastry, D. Goldberg, and G. Kendall. Genetic algorithms. In E. K. Burke and G. Kendall (Eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Chapter 4, pp. 97-125, Springer, 2005.
23. R. Lewis and B. Paechter. Application of the Grouping Genetic Algorithm to University Course Timetabling. *Proc. of the 5th European Conf. on Evol. Comput. in Combinatorial Optimization (EvoCOP 2005)*, LNCS 3448, pp. 144-153, 2005.
24. A. Acan and Y. Tekol. Chromosome Reuse in Genetic Algorithms. *Proc. of the 2003 Genetic and Evolutionary Computation Conference (GECCO 2003)*, pp. 695-705, 2003.
25. S. Louis and G. Li. Augmenting genetic algorithms with memory to solve traveling salesman problem. *Proc. of the 1997 Joint Conference on Information Sciences*, pp. 108-111, 1997.
26. A. Acan. An External Memory Implementation in Ant Colony Optimization. *Proc. of the 4th Int. Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS 2004)*, pp. 73-82, 2004.

Scheduling Semiconductor Manufacturing Operations: Problems, Solution Techniques, and Future Challenges

Lars Mönch • John W. Fowler • Stéphane Dauzère-Pérès • Scott J. Mason • Oliver Rose

Abstract In this paper, we discuss scheduling problems in semiconductor manufacturing. Starting from describing the manufacturing process, we identify typical scheduling problems that can be found in semiconductor manufacturing systems. We describe batch scheduling problems, job shop scheduling problems, scheduling problems with secondary resources, multiple orders per job scheduling problems, and scheduling problems related to cluster tools. We also present important solution techniques that are used to solve these scheduling problems by means of specific examples, and report on known implementations. Finally, we summarize some challenges in scheduling semiconductor manufacturing operations.

1 Introduction

Recently, the electronics industry has become the largest industry in the world. A key aspect of this industry is the manufacturing of integrated circuits. In semiconductor manufacturing, integrated circuits are produced on silicon wafers. In the past, efforts for reducing costs included decreasing the size of the chips, increasing the wafer sizes, and improving the yield, while simultaneously trying to improve operational processes inside the semiconductor manufacturing systems. Currently, it seems that the improvement of operational processes creates the best opportunity to realize the necessary cost reductions.

Semiconductor manufacturing is very capital intensive. Lots (denoted as jobs throughout the rest of this paper to conform with the scheduling literature) are the moving entities in semiconductor manufacturing systems. Each job contains a fixed number of wafers. The process conditions are very complex. We have to deal with parallel machines (also referred to as tools), different types of processes (batch and serial), sequence-dependent setup times, prescribed customer due dates for the jobs, and re-entrant process flows. Very often, we also have to cope with a large number of different products and a product mix that changes over time. Semiconductor manufacturing systems are prototypical examples for complex job shops.

Lars Mönch
University of Hagen
E-mail: Lars.Moench@fernuni-hagen.de

John W. Fowler
Arizona State University, Tempe
E-mail: John.Fowler@asu.edu

Stéphane Dauzère-Pérès
Ecole des Mines de Saint-Etienne
E-mail: Dauzere-Peres@emse.fr

Scott J. Mason
University of Arkansas, Fayetteville
E-mail: mason@uark.edu

Oliver Rose
Dresden University of Technology
E-mail: oliver.rose@tu-dresden.de

Crucial factors of competitiveness in semiconductor manufacturing are the ability to rapidly incorporate advanced technologies in electronic products, ongoing improvement of manufacturing processes, and, last but not least, the capability of meeting due dates for optimal customer satisfaction. In a situation where prices as well as the state of technology have settled at a certain level, the capability of meeting due dates along with the reduction of cycle time probably has become the most decisive factor to stand the fierce competition in the global market place. Cycle time is defined in this context as the time a job of wafers needs to travel through the semiconductor wafer manufacturing process including queue time, processing time, and transit time. Consequently, operations managers are under increasing pressure to ensure short and predictable cycle times.

Production control strategies based on dispatching rules are very popular in semiconductor manufacturing (cf. the recent survey [55]). But at the same time, scheduling approaches attracted researchers and people from industry working in semiconductor manufacturing for the last two decades (cf. [5], [58] for some older references). This is mainly because of a relatively high degree of automation (compared to other industries) that even 20 years ago allowed for automated real-time data collection and because of the fact that manual production control often leads to strange or unexpected behavior of the systems due to the complexity of the manufacturing process. The well-known book on intelligent scheduling [61] describes several large-scale scheduling prototypes from the semiconductor manufacturing domain. Until recently, full factory scheduling methods seemed to be too costly in comparison to dispatching methods. However, with the recent dramatic increase in computer efficiency full fab scheduling methods have become more competitive.

Because of the increasing automation pressure caused by automated material handling systems (AMHS) and new requirements on production control, it seems that scheduling approaches are both promising and necessary in the semiconductor manufacturing domain (cf. [43] for a survey related to scheduling requirements in semiconductor manufacturing). There is a lot of material from academia and also from software vendors related to scheduling in wafer fabs. However, except for the survey paper [53] from the early 1990s and a more current paper [30], there is no current source that reports on the breadth of scheduling problems within semiconductor manufacturing in one place.

All the present authors worked in the past and will continue work in the future on different aspects of scheduling for semiconductor manufacturing. In this paper, we will give a concise summary of the different problems and solution techniques as well as comments on problems that need more research efforts in the future. We note that this paper is not a complete review of the vast literature, however we hope to stimulate even more research and some successful real-world implementations in scheduling wafer fabs.

The paper is organized as follows. In Section 2, we describe the semiconductor wafer fabrication process briefly. We identify important scheduling problems in Section 3 and discuss solution techniques related to these scheduling problems. Finally, we identify some challenges for future research in Section 4.

2 Semiconductor Manufacturing Process Description

A semiconductor chip is a highly miniaturized, integrated electronic circuit consisting of thousands of components. Every semiconductor manufacturing process starts with raw wafers, thin discs made of silicon or gallium arsenide. Depending on the diameter of the wafer, up to a few hundred identical chips can be made on each wafer, building up the electronic circuits layer by layer in a wafer fab. Next, the wafers are sent to Probe, where electrical tests identify any individual die that is not likely to be good when packaged. The bad dice are either physically marked or an electronic map is made of them so that they will not be put in a package. The probed wafers are sent to an Assembly facility where the "good" dice are put into the appropriate package. Finally, the packaged dice are sent to a Test facility where they are tested in order to ensure that only good products are sent to customers. Wafer fab and Probe are often called the "Frontend" and Assembly and Test are often called the "Backend."

Considering the scale of integration, the type of chip, the type of package, and customer specifications, the whole manufacturing process may require up to 700 single processing steps and up to three months to produce. The four main stages of semiconductor manufacturing are shown in Figure 1.

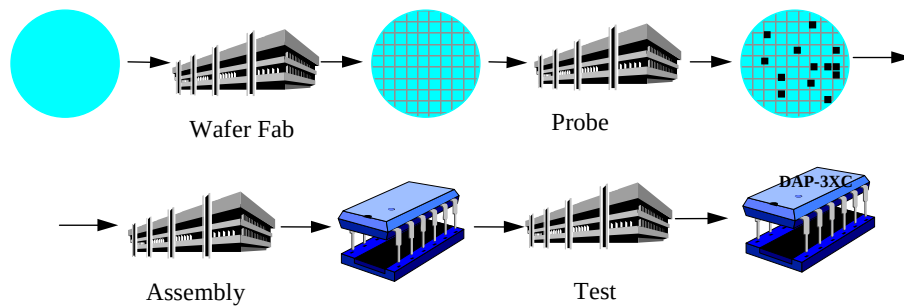


Figure 1: Main Stages of Semiconductor Manufacturing

Many authors have remarked on the difficulties of semiconductor manufacturing (cf. [2], [42], [50], [53], [55], [58] amongst others). Wafer fabrication has a number of unusual facets that are described below.

In a typical wafer fab, there often are dozens of process flows. Each process flow contains 200-700 processing steps and more than one hundred machines. These machines are expensive, ranging in price from a couple of hundred thousand dollars to over twenty million dollars per tool. The economic necessity to reduce capital spending dictates that such expensive machines be shared by all jobs requiring the particular processing operation provided by the machine, even though they may be at different stages of their manufacturing cycle. This results in a manufacturing environment that is different in several ways from both traditional flow shops as well as job shops. The main consequence of the re-entrant flow nature is that wafers at different stages in their manufacturing cycle have to compete with each other for the same machines. The typical re-entrant flow of a wafer fab is shown in Figure 2.

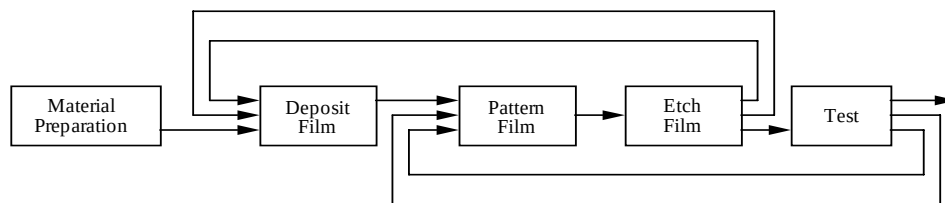


Figure 2: Re-entrant Flow in Wafer Fabs

Furthermore, the nature and duration of the various operations in a semiconductor flow differ significantly. Some operations require 15 minutes or less to process a job, while others may require over 12 hours. Many of these long operations involve batch processes. In reality, it is not uncommon for one-third of the fab operations to be batch operations. Batch machines tend to off-load multiple jobs (1 to 12) onto machines that are capable of processing only one job at a time. This leads to the formation of long queues in front of these serial machines and ultimately a non-linear flow of products in the factory. The probabilistic occurrence of long tool failures results in large variability in the time a job spends in process. High variability in cycle times prevents accurate prediction of production cycle times, resulting in longer lead-time commitments. There are some machines, such as implanters, that require significant sequence-dependent setups. If not scheduled well, these tools can become bottlenecks. Finally, some processing steps require an auxiliary resource (a so-called secondary resource), such as a reticle in photolithography, in order to process the job. Some of these auxiliary resources are quite expensive, so only a very limited number of them are purchased. Therefore, the challenge is to ensure that the machine and the auxiliary resource are available at the same time.

In modern 300-mm wafer fabs, wafers are transported in front-opening unified pods (FOUP) using an AMHS. Because of the increase in area and weight of the wafers, it is obvious that a manual handling of the wafers has to be eliminated. This fact increases the need for scheduling approaches. Automated material handling is always a critical operation in modern wafer fabs (cf. [1], [23], [33] for more details on AMHS in semiconductor manufacturing).

Next, we discuss backend operations. There are usually more types of parts being made in an assembly factory than in a wafer fab, but each part type requires 10-20 steps instead of 200-700. One difficulty in modeling these operations is the fact that a job is often divided into sub-jobs with each sub-job being sent to the next machine when it completes an operation. Thus, one job may be being processed across several machines at the same time. Another difficulty is that there is often a very significant amount of setup required to change over from one product type to another. Finally, batching machines are also often present in assembly factories. Test operations have several problems that are difficult to model. First, the sequence of test operations and the test times are not always fixed. These can be changed based on recent product yields, maturity of a product, and such. Second, there are two major types of equipment used in test operations. These are the test system itself (tester) and the loading mechanism (handler). The tester may have a single or multiple test heads connected to it. The interactions between the tester, the test heads, and the handler can be quite complex to model accurately. Finally, there can be significant sequence-dependent changeover times.

In this paper, we will restrict ourselves due to space limitations mainly to the wafer fab part of semiconductor manufacturing. But in a few situations, we will also discuss scheduling problems related to the backend stage.

3 Scheduling Problems and Solution Techniques

3.1 Batching Problems

A batch is defined as a group of jobs that have to be processed jointly. The completion time of a batch is determined as the completion time of the last job in the batch. A batch scheduling problem consists in grouping the jobs on each machine into batches and in scheduling these batches. Two types of batching problems are considered. The first type is called s-batching. Here, the processing time of a batch is the sum of the processing times of all jobs that form the batch. The second type is p-batching. In this case, the processing time of the batch is given by the maximum processing time of jobs contained in the batch (cf. [45], [30] for recent surveys related to batching in general and to batching for semiconductor manufacturing respectively).

In semiconductor manufacturing, there are some situations where s-batching is important. For example, several scheduling problems related to steppers in the photolithography area lead to s-batching problems. Here, runs are formed. A run is a group of jobs of the same mask layer. Therefore, the same reticle is required for processing. Forming a run and processing the jobs of the run in a consecutive manner avoids frequent reticle changes.

But p-batching is much more important in semiconductor manufacturing. The common assumption is that there exists a fixed batch size B as capacity of the batch machine. Burn-in ovens for semiconductor manufacturing are used to heat-stress test IC chips. Several chips can be tested in a burn-in oven simultaneously, so that a burn-in oven is a batch processing machine. The processing time of each batch is determined by the longest job processing time among those of all the jobs contained in the batch. There are many papers that deal with burn-in ovens, for example, the paper [56], where a dynamic programming formulation is coupled with a random key genetic algorithm. Simulated annealing-based heuristics are suggested in [31] for a burn-in scheduling problem.

The diffusion furnaces in wafer fabs are an example of batch machines. Here, the jobs are assigned to incompatible job families. While several jobs can be processed at the same time, jobs of different families cannot be processed together due to the chemical nature of the

process. The processing time of all jobs within one family is the same. Therefore, batching jobs with incompatible families is a special case of p-batching. We refer to [3], where the problem $Pm|batch, incompatible|TWT$ is discussed. We denote by *batch, incompatible* the p-batching with incompatible families, while the performance measure of interest is minimizing total weighted tardiness $TWT := \sum w_j \max(C_j - d_j, 0)$. The notation w_j is used for the weight of job j , C_j is used for the completion time, and d_j for the due date of job j . [3] extends solution techniques for $1|batch, incompatible|\sum T_j$ presented in [32]. Dispatching rules are used to form batches. Genetic algorithms assign the batches to machines and sequence them. This approach is extended to $Pm|r_j, batch, incompatible|\sum w_j T_j$ in [35].

This problem is harder because the given release dates r_j of the jobs require a decision whether it makes sense to wait for future jobs arrivals in case of incomplete batches or to start a non-full batch. Machine learning techniques applied to batching problems in semiconductor manufacturing are considered in [20], [36]. The problem $Pm|r_j, batch, incompatible|L_{max}$ where $L_{max} := \max\{C_j - d_j | j = 1, \dots, n\}$ denoting the maximum lateness, is solved in [27] using genetic algorithms.

3.2 Job Shop Problems

A wafer fab can be modeled as a complex job shop [28], [42]. This type of job shop contains unrelated parallel machines with sequence-dependent setup times and dedications, parallel batch machines, re-entrant flows, and ready times of the jobs. Using the $\alpha|\beta|\gamma$ notation these problems can be expressed by

$$FJ|r_j, s_{jr}, batch, incompatible, recrc|\sum w_j T_j, \quad (1)$$

where *FJ* denotes the flexible job shop, s_{kj} refers to sequence-dependent setup times, and the term *recrc* is used for the re-entrant flows. Alternatively, the scheduling problem can be modeled as

$$FJ|r_j, s_{jr}, batch, incompatible, recrc|L_{max}. \quad (2)$$

Problems of type (1) and (2) have been studied very intensively over the last ten years. Most of the solution approaches are based on disjunctive graphs [7]. A good documentation of attempts to solve problem (2) for test facilities can be found in [42]. Several variants of the shifting bottleneck heuristic are used. This heuristic decomposes the job shop scheduling problem into a set of smaller scheduling problems related to parallel machines. These smaller scheduling problems are called sub problems. Several aspects, like identifying appropriate sub problem solution procedures and determining an appropriate sequence to solve the sub problems, are discussed in a series of papers by Uzsoy and his group (cf., for example, [14], [15], [54]). Mason *et al.* suggested a modified shifting bottleneck heuristic to tackle problem (1) for wafer fabs in [28]. The performance of this scheme was assessed within a rolling horizon setting using discrete event simulation in [37]. Sub problem solution procedures based on genetic algorithms are suggested. A distributed variant of the shifting bottleneck heuristic for wafer fabs using information from an upper planning layer is discussed in [34]. This distributed

scheduling scheme is the base for the hierarchically organized multi agent system FABMAS [38]. While very often only single instances were considered, [49] contains the results of a rolling horizon scheme for problem (2). Later, several attempts were made to reduce the number of nodes in the scheduling graph by considering only specific bottleneck machines explicitly. Experiments with this approach for jobs shops that include machine breakdowns are presented in [4], [51], [52]. Only little is known on the impact of rescheduling techniques for complex job shops [29]. A multi-criteria approach for scheduling wafer fabs based on the shifting bottleneck heuristic is suggested in [44].

While some progress was made related to the integration of transportation operations and other generalizations in scheduling research for job shops (cf. [7] for a documentation of state of the art approaches), it seems that only some first steps into this direction were done in semiconductor manufacturing (cf. [16], [46]). Even the simulation of the base system for manufacturing and for transportation is sophisticated in the case of wafer fabs. But this is a requirement for assessing the performance of rolling horizon schemes.

Many current scheduling techniques for large-scale job shops with a makespan objective are based on metaheuristic search using the disjunctive graph formulation [7]. Because of the reentrant flows and different objectives, only initial steps are known to apply such methods to scheduling problems in semiconductor manufacturing [60]. Simulated annealing techniques based on a generalized disjunctive formulation for several batch machines and up-and-down stream machines are discussed in [59]. The resulting scheduling software called Batch Optimization Solver (BOS) is designed to solve real-world scheduling problems in the diffusion area of a wafer fab.

Beside the global scheduling problems (1) and (2), there are many papers that discuss scheduling approaches for specific machine environments in semiconductor manufacturing. We refer, for example, to [11] where a genetic algorithm is used to schedule jobs on furnace machines including sophisticated process conditions. Another example is given by [48] where ant colony optimization approaches are used to schedule jobs on a group of machines that is the leading bottleneck in a wafer fab.

The use of mixed integer and constraint programming approaches for solving real-world scheduling problems in a wafer fab is described in [6], [19]. Instead of considering the entire scheduling problem, a decomposition into different sub problems is performed before using mathematical programming tools.

3.3 Problems with Secondary Resources

As already described in Section 2, secondary resources are an important restriction in semiconductor manufacturing. Secondary resources are typically related to parallel machines. Photolithography steppers that require product and mask layer specific reticles are a typical example for such type of machines. Scheduling heuristics based on dispatching rules for the problem $Pm|r_j, aux|\sum w_j C_j$ are described, for example, in [12]. Here, we denote by *aux* the secondary resource. Several heuristics for stepper scheduling based on appropriate modifications of the Apparent Tardiness Cost (ATC) dispatching rule are described in [13]. The scheduling of a single batch processing machine with job families motivated by burn-in operations in a test facility is discussed in [24]. Load boards are considered as secondary constraints. Several heuristics are presented.

3.4 Multiple Orders per Job Problems

The combination of decreased line widths and more area per wafer in 300-mm wafer fabs result in fewer wafers being needed to fill an IC order of a customer. Each wafer fab will have only a limited number of FOUPs as they are expensive. A large number of FOUPs have the potential to cause overload in the AMHS. In addition, some batching tools have the same

processing times regardless of the number of wafers in the batch. Thus it is not reasonable to assign an individual FOUP to each order.

Therefore, 300-mm manufacturers often have the need and the incentive to group orders from different customers into one or more FOUPS to form production jobs. These jobs have to be scheduled on the various types of tool groups in the wafer fab and processed together. This class of integrated job formation and scheduling problems are called multiple orders per job (MOJ) scheduling problems.

There is a series of papers from Mason and his group that deal with single and parallel machine MOJ scheduling problems [21], [22], [47]. Metaheuristics, dispatching rules, and column generation techniques are used to solve the MOJ scheduling problems. The special case of a two-machine flow shop MOJ scheduling problem is presented in [25].

3.5 Scheduling of Cluster Tools

Cluster tools combine single-wafer processing modules with wafer handling robots in one closed environment (cf. [26] for a recent survey of cluster tool scheduling). Scheduling of cluster tools is challenging because the cycle time of wafers in a cluster tool depends on the used wafer recipes, cluster tool control and architecture, wafer waiting times, and sequencing [17]. A discrete event simulation tool is used in [17] to determine appropriate cycle times for wafers within cluster tools. Several techniques based, for example, on neural networks and beam search are suggested in [39], [40], and [41]. A simulated annealing approach is discussed in [57].

4 Future Challenges

There are several directions for future research related to scheduling in semiconductor manufacturing. The usage of modern metaheuristics for scheduling wafer fabs globally has to be researched. Furthermore, the impact of rescheduling strategies has to be investigated in much more detail. Future research requires a better incorporation of AMHS decisions into the job scheduling decision-making process. Algorithms are expected that take various specific properties of modern AMHS into account. It seems that a direct extension of the results presented for related problems in [7] is not straightforward.

There is a definite need to better understand the relationship between planning and scheduling decisions for complex manufacturing systems like wafer fabs. Some promising initial steps towards reaching this goal are described in [9]. Here, the interaction of more global and local scheduling decisions is studied. Based on the experiences with the FABMAS multi agent system [38] and the distributed shifting bottleneck heuristic [34], a hierarchical decomposition of wafer fab scheduling problems offers some advantage. However, more theoretical insights are necessary.

So far, only little is known on the interaction of Advanced Process Control (APC) and scheduling decisions. However, APC information should definitely impact scheduling constraints and criteria. For instance, scheduling some lots with the same process on a tool might be prioritized based on the requirements for the tool to get data on the process.

Finally, very often we find it hard to transfer scheduling results from an academic environment to the shop floor. This is mainly caused by data problems. Therefore, appropriate software representations of our scheduling algorithms are required that can better deal with distributed and with missing data.

References

1. Agrawal, G. K., Heragu, S. S., A Survey of Automated Material Handling Systems in 300-mm Semiconductor Fabs, IEEE Transactions on Semiconductor Manufacturing, 19(1), 112-120, (2006)

2. Atherton, L., Atherton, R. W., Wafer Fabrication: Factory Performance and Analysis. Kluwer Academic Publishers, Boston, Dordrecht, London, (1995)
3. Balasubramanian, H., Mönch, L., Fowler, J. W., M. Pfund, Genetic Algorithm Based Scheduling of Parallel Batch Machines with Incompatible Job Families to Minimize Total Weighted Tardiness, *International Journal of Production Research*, 42(8), 1621-1638, (2004)
4. Barua, A., Narasimhan, R., Upasani, A., Uzsoy, R., Implementing Global Factory Schedules in the Face of Stochastic Disruptions, *International Journal of Production Research*, 43(4), 94-109, (2005)
5. Bitran, G. R., Tirupati, D., Development and Implementation of a Scheduling System for a Wafer Fabrication Facility, *Operations Research*, 36(3), 377-395, (1988)
6. Bixby, R., Burda, R., Miller, D., Short-Interval Detailed Production Scheduling in 300mm semiconductor manufacturing using Mixed Integer Programming, *Advanced Semiconductor Manufacturing Conference*, 148-154, (2006)
7. Brucker, P., Knust, S. *Complex Scheduling*, Springer, Berlin, Heidelberg, (2006)
8. Brucker, P., Gladky, A., Hoogeveen, H., Kovalyov, M. Y., Potts, C. N., Tautenhahn, T., van de Velde, S., Scheduling a Batching Machine, *Journal of Scheduling*, 1, 31-54, (1998)
9. Bureau, M., Dauzere-Peres, S., Yugma, C., Vermarien, L., Maria, J.-B., Simulation Results and Formalism for Global-Local Scheduling in Semiconductor Manufacturing, *Proceedings of the 2007 Winter Simulation Conference*, 1768-1773, (2007)
10. Chandra, S. M., Mathirajan, M., Gopinath, R., Sivakumar, A. I., Tabu Search Methods for Scheduling a Burn-in Oven with Non-identical Job Sizes and Secondary Resource Constraints, *International Journal of Operational Research*, 3(1-2), 119-139, (2008)
11. Chien, C.-F., Chen, C.-H., A Novel Timetabling Algorithm for a Furnace Process for Semiconductor Fabrication with Constrained Waiting and Frequency-Based Setups, *OR Spectrum* 29, 391-419, (2007)
12. Cakiki, E., Mason, S. J. Parallel Machine Scheduling Subject to a Auxiliary Resource Constraints, *Production Planning and Control*, 18(3), 217-225, (2007)
13. de Diaz, S.L.M., Fowler, J. W., Pfund, M.E., Mackulak, G.T., Hickie, M. Evaluating the Impacts of Reticle Requirements in Semiconductor Wafer Fabrication, *IEEE Transactions on Semiconductor Manufacturing*, 18(4), 622- 632, (2005)
14. Demirkol, E., Mehta, S., Uzsoy, R., A Computational Study of Shifting Bottleneck Procedures for Shop Scheduling Problems, *Journal of Heuristics*, 3, 111-137, (1997)
15. Demirkol, E., Uzsoy, R. Decomposition Methods for Reentrant Flow Shops with Sequence Dependent Setup-Times, *Journal of Scheduling*, 3, 155-177, (2000)
16. Driessel, R., Mönch, L. An Integrated Scheduling and Automated Material Handling Approach for Complex Manufacturing Systems, *Proceedings of the 2008 IEEE International Conference on Industrial Engineering and Engineering Management*, (2008)
17. Dümmler, M. Using Simulation and Genetic Algorithms to Improve Cluster Tool Performance, *Proceedings of the 1999 Winter Simulation Conference*, 875-879, (1999)
18. Erramilli, V., Mason, S. J., Multiple Orders per Job Compatible Batch Scheduling, *IEEE Transactions on Electronics Packaging Manufacturing*, 29 (4), 285-296, (2006)
19. Fordyce, K., Bixby, R., Burda, R., Technology that Upsets the Social Order – A Paradigm Shift in Assigning Lots to Tools in a Wafer Fabricator – The Transition from Rules to Optimization, *Proceedings of the 2008 Winter Simulation Conference*, 2277- 2285, (2008)
20. Geiger, C.D., Uzsoy, R., Learning Effective Sequencing Rules for Batch Processor Scheduling, *International Journal of Production Research*, 46, 1431-1454, (2008)
21. Jampani, J., Mason, S. J., Column Generation Heuristics for Multiple Machine, Multiple Orders per Job Scheduling Problems, *Annals of Operations Research*, 159(1), 261-273, (2008)
22. Jia, J., Mason, S. J., Semiconductor Manufacturing Scheduling of Jobs Containing Multiple Orders on Identical Parallel Machines, *International Journal of Production Research*, To appear, (2009).

23. Jimenez, J., Mackulak, G.T., Fowler, J.W., Levels of Capacity and Material Handling System Modeling for Factory Integration Decision Making in Semiconductor Wafer Fabs, *IEEE Transactions on Semiconductor Manufacturing*, 21(4), 600-613, (2008)
24. Kempf, K., Uzsoy, R., Wang, C., Scheduling a Single Batch Processing Machine with Secondary Resource Constraints, *Journal of Manufacturing Systems*, 17(1), 37-51, (1998)
25. Laub, J. D., John W. Fowler, Keha, A. B., Minimizing Makespan with Multiple-Orders-per-Job in a Two-Machine Flowshop, *European Journal of Operational Research*, 128(1), 63-79, (2007)
26. Lee, T.-E. A Review of Scheduling Theory and Methods for Semiconductor Manufacturing Cluster Tools, *Proceedings of the 2008 Winter Simulation Conference*, 2179-2135, (2008)
27. Malve, S., Uzsoy, R. A Genetic Algorithm for Minimizing Maximum Lateness on Parallel Identical Batch Processing Machines with Dynamic Job Arrivals and Incompatible Job Families, *Computers & Operations Research*, 34(10), 3016-3028, (2007)
28. Mason, S. J., Fowler, J. W., Carlyle, W. M., A Modified Shifting Bottleneck Heuristic for Minimizing Total Weighted Tardiness in Complex Job Shops, *Journal of Scheduling*, 5 (3), 247-262, (2002)
29. Mason, S. J., Jin, S., Wessels, M. C., Rescheduling Strategies for Minimizing Total Weighted Tardiness in Complex Job Shops, *International Journal of Production Research*, 42(3), 613-628, (2004).
30. Mathirajan, M., Sivakumar, A. I. A., Literature Review, Classification and Simple Meta-analysis on Scheduling of Batch Processors in Semiconductor, *International Journal of Advanced Manufacturing Technology*, 29, 990-1001, (2006)
31. Mathirajan, M., Sivakumar, A. I., Kalaivani, P., An Efficient Simulated Annealing Algorithm for Scheduling Burn-in Oven with Non-identical Job Sizes, *International Journal of Applied Management and Technology*, 2(2), 117-138, (2004)
32. Mehta, S. V., Uzsoy, R., Minimizing Total Tardiness on a Batch Processing Machine with Incompatible Job Families, *IIE Transactions*, 30, 165-178, (1998)
33. Montoya-Torres, J. R., A Literature Survey on the Design Approaches and Operational Issues of Automated Wafer-transport Systems for Wafer Fabs, *Production Planning & Control*, 17(6), 648-663, (2006)
34. Mönch, L., Driessel, R., A Distributed Shifting Bottleneck Heuristic for Complex Job Shops. *Computers & Industrial Engineering*, 49, 673-680, (2005)
35. Mönch, L., Balasubramanian, H., Fowler, J. W., Pfund, M. E., Heuristic Scheduling of Jobs on Parallel Batch Machines with Incompatible Job Families and Unequal Ready Times, *Computers & Operations Research* 32, 2731-2750, (2005)
36. Mönch, L., Zimmermann, J., Otto, P., Machine Learning Techniques for Scheduling Jobs with Incompatible Families and Unequal Ready Times on Parallel Batch Machines, *Journal of Engineering Applications of Artificial Intelligence*, 19(3), 235-245, (2006)
37. Mönch, L., Schabacker, R., Pabst, D., Fowler, J. W., Genetic Algorithm-based Subproblem Solution procedures for a Modified Shifting Bottleneck Heuristic for Complex Job Shops, *European Journal of Operational Research*, 177(3), 2100-2118, (2007)
38. Mönch, L., Stehli, M., Zimmermann, J., Habenicht, I., The FABMAS Multi-Agent-System Prototype for Production Control of Waferfabs: Design, Implementation, and Performance Assessment, *Production Planning & Control*, 17(7), 701-716, (2006)
39. Niedermayer, H., Rose, O., Solution Approaches for the Cluster Tool Scheduling Problem in Semiconductor Manufacturing, *Proceedings of the 16th European Simulation Symposium*, (2004)
40. Niedermayer, H., Rose, O., Approximation of the Cycle Time of Cluster Tools in Semiconductor Manufacturing, *Proceedings of the 2004 Industrial Engineering Research Conference*, (2004)
41. Oechsner, S., Rose, O., Scheduling Cluster Tools Using Filtered Beam Search and Recipe Comparison, *Proceedings of the 2005 Winter Simulation Conference*, (2005)

42. Ovacik, I. M., Uzsoy, R., *Decomposition Methods for Complex Factory Scheduling Problems*, Kluwer Academic Publishers, Massachusetts, (1997)
43. Pfund, M., Mason, S., Fowler, J. W., *Dispatching and Scheduling in Semiconductor Manufacturing*, *Handbook of Production Scheduling*, J. Herrmann, (eds.), Springer, Heidelberg, 213-241, (2006)
44. Pfund, M. E., Balasubramanian, H., Fowler, J. W., Mason, S. J., Rose, O., *A Multi-Criteria Approach for Scheduling Semiconductor Wafer Fabrication Facilities*, *Journal of Scheduling*, 11(1), 29-47, (2008)
45. Potts, C. N., Kovalyov, M. Y., *Scheduling with Batching: a Review*, *European Journal of Operational Research*, 120, 228-249, (2000)
46. Qu, P., Steinmiller, B., Mason, S. J., *Incorporating Automated Material Handling Systems Into a Disjunctive Graph*, *Proceedings of the 2004 Industrial Engineering Research Conference*, Houston, (2004)
47. Qu, P., Mason, S. J., *Metaheuristic Scheduling of 300mm Jobs Containing Multiple Orders*, *IEEE Transactions on Semiconductor Manufacturing*, 18 (4), 633-643, (2005)
48. Song, Y., Zhang, M. T., Yi, J., Zhang, L., Zheng, L., *Bottleneck Station Scheduling in Semiconductor Assembly and Test Manufacturing using Ant Colony Optimization*, *IEEE Transactions on Automation Science and Engineering*, 4(4), 569 – 578, (2007)
49. Souriraja, K., Uzsoy, R., *Hybrid Decomposition Heuristics for Solving Large-Scale Scheduling Problems in Semiconductor Wafer Fabrication*, *Journal of Scheduling*, 10, 41-65, (2007)
50. Sze, S. M., *Semiconductor Devices: Physics and Technology*, John Wiley & Sons, New York, Second Edition, (2001)
51. Upasani, A. A., Uzsoy, R., Sourirajan, K., *A Problem Reduction Approach for Scheduling Semiconductor Wafer Fabrication Facilities*, *IEEE Transactions on Semiconductor Manufacturing*, 19(2), 216-225, (2006)
52. Upasani, A. A., Uzsoy, R., *Integrating a Decomposition Procedure with Problem Reduction for Factory Scheduling with Disruptions: a Simulation Study*, *International Journal of Production Research*, 46 (21), 5883-5905, (2008)
53. Uzsoy, R., Lee, C. Y., Martin-Vega, L., *A Review of Production Planning and Scheduling Models in the Semiconductor Industry, Part II: Shop Floor Control*, *IIE Transactions*, 26, 44-55, (1994)
54. Uzsoy, R., Wang, C.-S., *Performance of Decomposition Procedures for Job-Shop Scheduling Problems with Bottleneck Machines*, *International Journal of Production Research*, 38, 1271-1286, (2000)
55. Varadarajan, A., Sarin, S. C., *A Survey of Dispatching Rules for Operational Control in Wafer Fabrication*, *Proceedings INCOM 2006*, (2006)
56. Wang, C. S., Uzsoy, R., *A Genetic Algorithm to Minimize Maximum Lateness on a Batch Processing Machine*, *Computers & Operations Research*, 29, 1621-1640, (2002)
57. Yim, S. J., Lee, D. Y., *Scheduling Cluster Tools in Wafer Fabrication Using Candidate List and Simulated Annealing*, *Journal of Intelligent Manufacturing*, 10, 531–540, (1999)
58. Wein, L., *Scheduling Semiconductor Wafer Fabrication*, *IEEE Transactions on Semiconductor Manufacturing*, 1, 115-130, (1988)
59. Yugma, C., Dauzere-Peres, S., Derreumaux, A., Sibille, O., *A Batch Optimization Software for Diffusion Area Scheduling in Semiconductor Manufacturing*, *Advanced Semiconductor Manufacturing Conference 2008 (ASMC 2008)*, 327-332, (2008)
60. Zoghby, J., Barnes, J. W., Hasenbein, J. J., *Modeling the Reentrant Job Shop Scheduling Problem with Set-ups for Metaheuristic Searches*, *European Journal of Operational Research*, 167, 336-348, (2005)
61. Zweben, M., Fox, M. S. *Intelligent Scheduling*, Morgan Kaufman Publishers, San Fransisco, (1994)

Hybrid of the weighted minimum slack and shortest processing time dispatching rules for the total weighted tardiness single machine scheduling problem with availability constraints

Ibrahim H. Osman · Hocine Belouadah · Krzysztof Fleszar · Maysaa Saffar

Abstract The paper addresses the total weighted tardiness single-machine scheduling problem with machine availability constraints. Each machine can handle one job at a time and has a set of maintenance periods, during which the machine must be stopped for service. Each maintenance period lasts for a fixed interval of time and starts after a fixed interval of processing non-resumable jobs. A non-resumable job must be processed and completed outside the maintenance periods. The single-machine periodic maintenance problems with a total weighted tardiness objective appear to be a new class of problems for which, in this paper, a mixed integer linear programming formulation and a new composite dispatching rule based on a combination of the weighted minimum slack and shortest processing time rules are introduced. New data set instances with different characteristics are generated using existing benchmarks from the literature. Extensive computational results are reported to testify the power of the proposed dispatching-based rule heuristic by comparing it to other heuristics inspired from the literature and developed for this new problem. Finally the paper concludes with a summary and highlights further research directions.

1 Introduction

Acronyms

TWT	total weighted tardiness
SMSP	single machine scheduling problem
MA	machine availability
PM	periodic maintenance
MILP	mixed integer linear programming
DR	dispatching rule
DRH	dispatching rule heuristic

The Total Weighted Tardiness (TWT) Single-Machine Scheduling Problem (SMSP) has been a very active research and practice areas for many decades based on a basic assumption that the machine is available at all times. In modern manufacturing and service industries the all-time availability assumption may not be justifiable; due to the owner's interest in maximizing the utilization of expensive and bottleneck machines. As a consequence, decision makers recognized the importance of introducing a periodic maintenance schedule to extend the life and availability of machine, to avoid breakdown and loss of services, and to reduce building-up of inventories. For instance, OTIS Company has designed an intelligent predictive system which records the working condition of an elevator to decide on maintenance visits in order to minimize elevator downtime. Such groundbreaking optimization system is one of OTIS competitive advantages to boost revenues and enhance reputations [8]. Furthermore, the

Ibrahim H. Osman, Krzysztof Fleszar &
Maysaa Saffar

Olayan School of Business, American University of Beirut, P.O.Box 11-0236, Beirut Lebanon
E-mail: {ibrahim.osman, maysaa.saffar, krzysztof.fleszar}@aub.edu.lb

Hocine Belouadah
Faculty of Economic and Commerce Sciences, University of M'Sila, Algeria.
E-mail: hbelouadah@yahoo.fr

importance of introducing preventive maintenance (PM) to increase machine availability (MA) is also highlighted in Raza *et al* [20] who emphasized that a sound PM in a manufacturing environment results in reducing corrective maintenance cost and defective production cost as well as increasing the availability of production facility. Finally, Cassady & Kutanoglu [3] stressed the practical importance of integrating PM planning and production scheduling for which they developed an integrated mathematical model to minimize the total weighted completion time on a single machine. They reported improvement with average savings ranging from 2 to 20% depending on the nature of the manufacturing system. Not to mention real life applications in the airline industry where there are special repair stations (virtual single-machine) to perform different types of maintenance repairs (A, B or C) with varying processing times on each aircraft. The repair stations are limited in number worldwide, and each station requires regular maintenance to increase its utility to meet its continuous demands of on-time maintenance of aircrafts, otherwise a heavy penalty would be incurred equal to at least the lost income from a non repaired aircraft. Another application in the airline industry would be an aircraft that must be serviced after a fixed total of flying mileage before it is allowed to resume operation; delays in maintenance of any flight would normally incur a huge penalty cost. It should be noted that machine availability constraints in general include maintenance, vacations, and breaks. In this paper, we limit our study to a set of maintenance periods during which the machine is not available. We denote the total weighted tardiness single machine scheduling problem with availability constraints as (TWT-SMSP-MA) and that with a periodic maintenance schedule by (TWT-SMSP-PM).

On the theoretical side, NP-hardness in the strong sense of TWT-SMSP was established by Lenstra, Rinnoy Kan and Brucker [14]. Adiri, Bruno, Frosting and Rinnoy Kan [1] considered the SMSP to minimize *total completion time* where the machine is unavailable during some intervals. They showed that SMSP is NP-hard when there is only one maintenance period in the schedule. Leon & Wu [15] introduced SMSP with the machine availability constraint to minimize the *maximum lateness* of jobs. They presented a nonlinear mixed integer model where the maintenance periods are treated as a set of additional jobs with flexible ready-times and artificial due dates. Lee [13] proved that SMSP-MA with one maintenance period to minimize the total tardiness is NP-hard. Consequently, TWT-SMSP-PM with m ($m > 1$) maintenance periods is also NP-hard since it is a generalization of both TWT-SMSP and SMSP-PM. Qi *et al.*[19] also proved that SMSP where jobs and maintenances are scheduled simultaneously to minimize the *total completion time* is NP-hard in the strong sense for which they developed three heuristics and a branch-and-bound algorithm for solving instances up to 20 jobs. Graves and Lee [10] presented SMSP with one maintenance activity to minimize the *total weighted completion time*.

Recently, Liao & Chen [16] considered SMSP-PM with several maintenance periods to minimize the *maximum tardiness* for which they developed a branch-and-bound method and a heuristic based on batching of jobs and earliest due date scheduling rule to solve instances up to 30 jobs. Chen [4] developed a heuristic and a branch-and-bound algorithm to minimize the *total flow time* in order to solve instances up to 30 jobs. Raza and Al-Turki [20] addressed the problem of joint scheduling of maintenance operations and jobs to minimize the *total completion time* to ensure high productivity. They discussed properties for an optimal schedule in order to develop tabu search and simulated annealing meta-heuristics to solve instances up to 40 jobs. Ji *et al.* [11] developed a construction heuristic with a worst-case ratio less than two to minimize the *makespan*. Chen [5] proposed a heuristic and a branch-and-bound algorithm to minimize separately the total *flow time* and the *maximum tardiness*. Chen [6] developed a heuristic as well as a branch-and-bound algorithm considered to minimize the *number of tardy jobs* and solved instances up to 32 jobs.

From the above review, it can be seen that SMSP-PM with the objective of minimizing *total weighted tardiness*, has not been addressed in the literature. Moreover, jobs with due dates are of great interest in business since the cost of tardiness incurs different types of penalties such as: dissatisfied customers with loss of future sales and goodwill, and rush shipping cost among others, Biskup and Feldmann [2]. Hence, our aim in this paper is to fill in

this gap in the literature and to provide new approximate and exact approaches for TWT-SMSP-PM based on its characteristics and enhanced versions of existing methods in the TWT-SMSP literature. The remaining part of the paper is organised as follows. In section 2, a statement, notation and mixed integer linear formulation for the TWT-SMSP-PM are introduced. Section 3, proposes the new composite heuristic with enhancement of existing dispatching rules. Section 4 presents our extensive computational experience using benchmark instances from the literature and additional maintenance periods with maintenance intervals of different characteristics. Section 5 terminates with a conclusion and further research directions.

2 Problem Statement, Notation and Mixed Integer Linear Formulation

The total weighted tardiness single-machine scheduling problem with periodic maintenance, TWT-SMSP-PM, has a number of assumptions that are stated as follows. On the job side, it is assumed that there exists a set of n jobs, $N=\{1, \dots, n\}$, each job $i \in N$ has a processing time p_i , a due date d_i , and a penalty w_i for each unit of time delayed beyond its due date; all p_i , d_i , and w_i are integer positive values. It is also assumed that a job is available for processing at time zero without pre-emption, i.e. once the job is started, it must be completed without any interruption. On the machine side, the existence of a set of m Maintenance Periods $MP = \{1, \dots, m\}$ is assumed. Each period is required after a fixed interval of T units of machine processing time. Each maintenance period is a fixed length of repair time (ML). Here, the objective is to determine a sequence/schedule of jobs on the machine to minimize the total weights of tardy jobs subject to the pre-determined schedule of maintenance periods. The following set of notations will be used in the remaining part of the paper:

i	Index of job i
p_i	Processing time of job i and $p_i < T$
d_i	Due date of job i
w_i	Penalty of job i
c_i	Completion time of job i
l_i	Lateness of job i , where $l_i = \text{Max}(0, c_i - d_i)$
s_i	Start of processing time of job i
n	Number of jobs
N	Set of jobs, $N = \{1, \dots, n\}$,
P	The total sum of processing of all jobs, $P = \sum_{i \in N} p_i$
AP	Average processing times of the Pending jobs
$\sigma(j)$	Job i in the j^{th} position of the solution sequence
B_k	Batch k for processing a subset of jobs within an interval of time T
T	Length of processing time for a batch
SL_k	Slack time of B_k or the time left after processing jobs in B_k
M_k	Maintenance period k
m	Number of maintenance periods
m_k	Starting time of the maintenance period k
ML	Length of time to perform the maintenance/repair
M	Very large positive integer
α	Look-ahead parameter for the apparent tardiness cost (ATC) dispatching rule
$\text{Max\_}L$	Maximum number of backtrack levels
R	Due Date Range
Tf	Tightness Factor

Figure 1 A sequence of jobs and periodic maintenances in a schedule.

B_l			M_l	...	B_k			M_k	...	B_m			M_m	B_{m+1}		
$\sigma(1)$	$\sigma(2)$	...	ML	...	...	$\sigma(j)$	...	ML	...	...	...	...	ML	...	$\sigma(n)$	

A sequence of jobs and maintenance periods in a given sequence solution S is typically represented by $S = \{B_l, M_l, \dots, B_k, M_k, \dots, B_m, M_m, B_{m+1}\}$ where $\sigma(j)$ indicates the order of job i in a batch B_k for $j = 1, \dots, n$ and $k = 1, \dots, m+1$ as depicted in Figure 1.

2.1 Mixed Integer Linear Programming Formulation

This sub-section proposes a mixed integer linear programming (MILP) formulation to the TWT-SMSP-PM. It is inspired from the work of Leon and Wu (1992) and is presented as follows. The objective function of minimizing the total weighted tardiness of jobs can be expressed by a non-linear term as follows: *Minimize* $NZ = \sum_{i \in N} w_i \times \text{Max}(0, l_i)$ (1)

Note that the tardiness (lateness) l_i of a job i is either $l_i = c_i - d_i > 0$ or $l_i \leq 0$. Hence, $\text{Max}(0, l_i)$ reduces to l_i in (2) with the addition of constraint (3) and the addition of non-negativity constraint $l_i \geq 0$ in (9). Consequently, the nonlinear objective (1) can be rewritten as a linear objective function Z as follows:

$$Z = \text{Minimize} \sum_{i \in N} w_i \times l_i \quad (2)$$

Subject to:

a) *Linearization of the objective function:*

$$c_i = s_i + p_i \quad \text{and} \quad l_i \geq c_i - d_i \quad \forall i \in N \quad (3)$$

b) *Logical relationships among jobs:* Either job i starts before job j or j starts before i :

$$c_i \leq s_j \quad \text{or} \quad c_j \leq s_i \quad \forall i, j \in N \quad \text{where } i < j \quad (4)$$

The logical relationship in equation (4) can be converted into linear forms at the expense of additional set of logical δ_{ij} variables which are defined as follows. Let

$$\delta_{ij} = \begin{cases} 1 & \text{if job } i \text{ precedes job } j \\ 0 & \text{otherwise} \end{cases} \quad \forall i < j \in N$$

Consequently, equation (4) can be rewritten using the following two additional constraints expressed in equation (5):

$$\begin{aligned} \text{i)} \quad & c_i \leq s_j + M(1 - \delta_{ij}) \quad \forall i, j \in N \quad i < j \\ \text{ii)} \quad & s_i + M\delta_{ij} \geq c_j \quad \forall i, j \in N \quad i < j \end{aligned} \quad (5)$$

c) *Logical relationships among jobs & maintenance periods:* Either job i starts before a maintenance k or k starts before i :

$$c_i \leq m_k \quad \text{or} \quad m_k + ml \leq s_i \quad \forall i \in N, \forall k \in MP \quad (6)$$

Similarly, equation (6) can be converted into linear forms as follows:

$$\gamma_{ik} = \begin{cases} 1 & \text{if job } i \text{ precedes maintenance } k \\ 0 & \text{otherwise} \end{cases} \quad \forall i \in N, k \in MP$$

$$\begin{aligned} \text{i)} \quad & c_i \leq m_k + M(1 - \gamma_{ik}) \quad \forall i \in N \& \quad k \in MP \\ \text{ii)} \quad & m_k + ml \leq s_i + M\gamma_{ik} \quad \forall i \in N \& \quad k \in MP \end{aligned} \quad (7)$$

d) *Setting each maintenance period to start at a constant time:*

Set each maintenance period to a known and prefixed constant future time in chronological order. They can be set as follows:

$$m_k = k * T + (k - 1) * ml \quad \forall k \in MP \quad (8)$$

e) *Non-negativity constraint:*

$$\delta_{ij}, \gamma_{ik} \in \{0,1\}, s_i \geq 0, m_k \geq 0, \text{ and } l_i \geq 0 \quad \forall i, j \in N, \forall k \in MP \quad (9)$$

In summary, the MILP would consist of the constraints expressed in equations 2, 3, 5, 7, 8, and 9. The number of maintenance periods can be estimated as follows: $m = \lceil P/T \rceil$ i.e. the total sum of processing of all jobs, P over T , is truncated upward. Since TWT-SMSP-PM is NP-Hard and only small sized instances can be solved to optimality, MILP is used to determine the largest size that can be solved to optimality and to access the effectiveness of solution quality and computational efficiency of the newly developed approximate heuristics.

3 Modified Weighted Minimum Slack & Shortest Processing Time Dispatching Heuristics

This section first introduces the most common dispatching rules in the total weighted tardiness single machine problem (TWT-SMSP) literature. Second, some of the greedy disadvantages of such dispatching rules are highlighted in order to develop a *push-back* theorem to overcome such weakness when applied to TWT-SMSP-PM. Third, a composite dispatching rule denoted by (WMSPT) is newly introduced. WMSPT is based on a combination of the *weighted minimum slack* (WMS) and the *weighted shortest processing time* (WSPT) rules. Each of the various dispatching rules is then embedded into a generic dispatching rule heuristic to produce approximate solutions for TWT-SMSP-PM. Finally, a backtracking with a limited binary enumeration strategy is proposed to correct the greedy and myopic ranking of jobs by a dispatching rule and to further enhance performance.

3.1 Dispatching Rules

Research in dispatching rules (DR) has been active for several decades and many different rules have been studied in the literature. DR is a priority function to select the next job for processing on the machine. It is based on attributes of a job such as weight, processing time, and due date. Such priority function may be classified as dynamic or static depending on whether the attributes are constant or time-dependent. In a time-dependent case, a job may have different priorities for processing on the machine at different points in time. Table 1 summarises the most well-known dispatching rules in the TWT-SMSP literature. Most of them are based on Emmons [9]'s dominance conditions and their enhancements for TWT-SMSP. Emmons' dominance properties established precedence relations among jobs in an optimal schedule. For instance, it is well known that the Earliest Due Date rule (EDD) gives an optimal sequence for the TWT-SMSP when all due dates are sufficiently loose or spread out, whereas the WSPT rule gives an optimal solution when either all due dates are zeros or all jobs are tardy. Under this case the TWT-SMSP reduces to the total weighted completion time which is known to be solved optimally by WSPT rule, Smith [21]. WSPT rule sequences jobs in non-increasing order of $\{w_i / p_i\}$. Therefore, it is natural to exploit such nice properties by combining such simple and efficient priority rules in various ways. For instance, Vopsalainon and Morton [24] developed the *apparent tardiness cost* (ATC) composite rule which processes jobs in non-increasing order of time-dependent index based on a combination of the WSPT rule and the weighted minimum slack (WMS) rule using an exponential function to alternate the DR selection of jobs. Morton and Pentico [17], Volgenant and Teerhuis [23] and Kanet and Li [12] confirmed the superiority of the ATC rule over EDD, WSPT and WRA rules. However, the ATC was comparable to the *modified weighted due date rule* (WMDD) introduced by Kanet and Li [12]. In this section, we are following the same principle of combining the same simple and efficient WSPT and WMS rules, however, using a *Max* function to alternate job selection. It should be mentioned that all the rules (R1-R6) are from the literature, we refer to the excellent book by Pinedo [18] and the recent paper of Kanet and

Li [12] for more details. However, WMSPT (R7) is our new dispatching rule to be explained in the next section.

Table 1: A set of dispatching-rules for constructing approximate sequences

	Rule	Definition	Priority Index	Type
R1	ATC	Apparent Tardiness Cost	$\max\left\{\frac{w_i}{p_i} \exp\left(\frac{-\max(d_i - p_i - t, 0)}{\alpha P}\right)\right\}$	Dynamic
R2	WEDD	Weighted Earliest Due Date	$\max\left\{\frac{w_i}{d_i}\right\}$	Static
R3	WMDD	Weighted Modified Due Date	$\min\left\{\frac{\max(p_i, d_i - t)}{w_i}\right\}$	Dynamic
R4	WMS	Weighted Minimum Slack	$\min\left\{\frac{\max(d_i - p_i - t, 0)}{w_i}\right\}$	Dynamic
R5	WSPT	Weighted Shortest Processing Time	$\min\left\{\frac{p_i}{w_i}\right\}$	Static
R6	WRA	Weighted Remaining Allowance	$\min\left\{\frac{d_i - t}{w_i}\right\}$	Dynamic
R7	WMSPT	Weighted Minimum Slack Shortest Processing time	$\min\left\{\frac{\max(d_i - p_i - t, p_i)}{w_i}\right\}$	Dynamic

3.2 The Proposed Weighted Minimum Slack & Shortest Processing Time Dispatching Rule (WMSPT) Heuristic

In this section, we first discuss our new proposed composite WMSPT dispatch rule. Second, we introduce a *push-back* theorem to propose a corrective strategy to fill in slack times created in a batch by any greedy dispatching rule due to the introduction of maintenance constraints. Finally, both the WMSPT rule and the push-back strategy are implemented into what is called a WMSPT heuristic.

3.2.1 WMSPT dispatching rule

Our WMSPT dispatching rule uses the same components of the ATC rule. However, WMSPT is a *parameter-free* approach which combines the two well-known simple and efficient priority rules, namely, the shortest processing time (WSPT) and the WMS rule. The WSPT rule processes the jobs in non-decreasing order of $\{p_i / w_i\}$ and it is optimal for the TWT-SMSP when all due dates are zeros, i.e. all jobs are tardy, Smith [21]. However, the WMS rule processes the jobs in non-decreasing order of their weighted slack times $\{\max(d_i - p_i - t, 0) / w_i\}$. The WMS rule is also optimal when all due dates are sufficiently spread out, Pinedo [18]. Our new proposed composite rule, which is denoted as WMSPT, attempts to maintain the optimality characteristics of the two dispatching Weighted Minimum Slack and Weighted Shortest Processing Time rules. Under WMSPT, jobs are scheduled one at a time: each time the machine becomes free, a priority ranking index is computed for the remaining unscheduled jobs. The job with the smallest index is then selected for processing next. The ranking index is time dependent and defined as follows:

$$I_i(t) = \left\{ \frac{\max(d_i - p_i - t, p_i)}{w_i} \right\} \quad (10)$$

Clearly WMSPT alternates between WMS and WSPT to capture as much as possible their optimality characteristics. The main advantage of WMSPT lies in its simplicity and the lack of any parameter to control or estimate as compared to ATC.

3.2.2 WMSPT heuristic

Generally for scheduling problems, a dispatching rule is normally embedded in what is called a dispatching rule heuristic (DRH). In this section, we shall explain a generic DRH for TWT-SMSP-PM. Initially, a dispatching rule is chosen to generate priority indices for processing unscheduled jobs in an empty schedule consisting of one empty batch available at time 0. The set of unscheduled jobs are then sorted in increasing or decreasing order in UN . DRH then selects the first job in UN and inserts it in the next available position provided it is *feasible*. A job is feasible for processing in the current processing batch if there is enough slack time. Whenever infeasibility is encountered, a new processing batch is created and the scheduling process is continued until all jobs are scheduled. Note that the only difference between static and dynamic dispatching rules is that in the dynamic case, the priority of selecting a job is updated after each insertion, whereas the priority remains unchanged under the static case. The operation of sorting jobs is therefore executed only once in the static case where it is needed after each update in the dynamic case. The DRH steps are presented as follows:

- Step 1. Initialize:** Given a set of jobs N ; select DR; set the following:
- Initial schedule $S = \phi$; unscheduled jobs $UN = N$; job position index in the solution sequence $j = 0$, and the current time $t = 0$.
 - the initial batch index $k = 0$
- Step 2. Create new batch:** $k = k + 1$, $B_k = \phi$, $SL_k = T$ (initial slack equals processing time in a batch)
- Step 3. Rank jobs:** Compute a priority for each unscheduled job in UN and order jobs in UN according to the DR ranking (static case only).
- Step 4. Select next job:** Select best job i^* in UN (first job in the static case).
If $p_{i^*} \leq SL_k$, **then** go to *Schedule Job* (Step 5)
Else set $t = m_k + ml$ then go to *Create new batch* (Step 2).
- Step 5. Schedule Job:** Insert job i^* in batch $B_k = B_k \cup \{i^*\}$; set $j = j + 1$ and insert job i^* immediately after the last position in the current sequence, $\sigma(j) = i^*$; update the processing time $t = t + p_{i^*}$ and the slack time $SL_k = SL_k - p_{i^*}$
- Step 6. Update list** by removing job i^* from $UN = UN \setminus \{i^*\}$, **if** $UN \neq \phi$ **then** go to *Select next job* (Step 4) in static case or *Rank jobs* (Step 3) in dynamic case.
- Step 7. Terminate:** Compute the total weighted tardiness associated with the sequence obtained in solution S .

Note that the above presented DRH is generic and useful when attempting to find a good schedule for the single machine scheduling without periodic maintenance with single objectives such as *makespan*, *total completion time*, or *maximum lateness*. However, the addition of the periodic maintenance constraint creates an additional complexity which makes DRH less effective. Mainly, if a job does not pass the feasibility test, i.e. fit into the remaining slack of the current batch, a new batch is created. As a result, a large slack time may result at the end of the previous batch. Such slack time may be enough to process a lower priority unselected job that can better fit in, instead of attempting to process the higher priority job in a

greedy fashion. Such fashion creates more wasted slack time. Therefore, sorting jobs on the basis of one priority rule may not lead to an acceptable schedule. The following pushback theorem is inspired by the previous observation and is used to devise a new enhancement to modify *Step 3* of DRH steps to overcome the above weakness.

Theorem (Pushback)

Given a solution $S = \{B_1, M_1, \dots, B_i, M_i, \dots, B_k, M_k, \dots, B_m, M_m, B_{m+1}\}$ of ordered batches, if there exists a job $j \in B_k$ such that job j is feasible to be processed in the slack time of an earlier batch B_c ($c < k$) then the new solution after the pushback move would always have a total weighted tardiness smaller than or equal to the original one.

Proof :

Let WTA_j and WTB_j be the weighted tardiness of job j after the push-back move into B_c and its earlier weighted tardiness in B_k , respectively. Let C_h be the completion time of the last job h in batch B_c with an available slack time SL_c , CB_j the completion time of job j in batch B_k , and CA_j the completion time after job j is processed after job h in B_c . After execution of the feasible move we get $B_c = B_c \cup \{j\}$ and $B_k = B_k \setminus \{j\}$. By definition, we have the following relationships:

- (I) $p_j \leq SL_c$ and $CA_j = C_h + p_j \leq m_c < m_c + ml$, where the last term is the starting time of the next batch B_{c+1} .
- (II) Since $c < k$ then the starting time and maintenance period (c) is less than or equal to the starting time of batch k i.e. $(m_{k-1} + m_l) \leq CB_j$.

Consequently from I and II, we have $CA_j < CB_j$. Taking away a positive term from both sides ($d_j > 0$), we get $CA_j - d_j < CB_j - d_j$, then $Max(CA_j - d_j, 0) < Max(CB_j - d_j, 0)$ i.e. we would always have the new lateness of job j in the new batch smaller than the lateness of job j in the original batch regardless of its weight. Hence, the proof is complete.

Based on the pushback theorem, the new WMSPT heuristic can be described to construct in single pass a TWT-SMSP-MA solution as follows. The WMPST heuristic follows exactly the same *Steps 1-6* of the generic DR heuristic, except when an infeasible job is encountered, a *push-back (PB) strategy* is applied. The *PB* strategy attempts to capture the benefit of bringing back low priority jobs to fill-in available slack times at the end of a current batch i.e., instead of creating a new batch, an attempt is made to select the next unscheduled feasible job that can fit into the remaining slack time. Therefore, the *WMSPT with PB strategy* requires a modification of only *Step 4* into *Step 4a* as follows:

Step 3a Push-Back Feasibility Check: Select job i^* for scheduling from the top of *UN* according to the WMSPT ranking index in equation (10).

- 3.1 **If** $p_{i^*} \leq SL_k$, **then** go to scheduling (Step 5),
- 3.2 **Else** attempt to select the next job (b) that can fit into the current slack time,
- 3.3 **If** there exists no such job, **then** set $t = m_k + ML$ and go to create new batch (step 2),
- 3.4 **Else** go to scheduling job (b) in Step 4.

Note that the pushback strategy is inspired from the pushback theorem and it can be embedded into any DR heuristic using DR in Table 1. The letter E will be added to the DRH name to denote that the DRH is applied with pushback for example EWMSPT denotes WMSPT with Pushback.

4 Computational Experience

In this section we report our computational experience with the various developed exact and approximate algorithms. All the algorithms are coded in C# and run on a Pentium 4 3.2 GHz personal computer. In the following subsection, the generation of benchmarking data test instances is explained. Next, the effectiveness in solution quality and efficiency in computation time expressed in CPU in *one thousand of a second (ms)* of the proposed dispatching heuristics are discussed with respect to the characteristics of the problem instances. Further, the impact of the pushback strategy on the solution quality is investigated.

4.1 Benchmarking Data Test Instances

The data test instances for the TWT-SMSP-PM are generated using the original data test instances that were generated for the TWT-SMSP by Crauwels, Potts, and Van Wassenhove [7] and are available in the OR-library. The characteristics of the original data test can be explained as follows. There are 125 test instances that were generated for each problem size n of 30, 40, 50 and 100 jobs. There are three information data files one for each size namely i30, i40, i50 and i100. Each file contains the test instance ID, the list of n jobs' ID, followed by their processing time p_i , weight w_i , and due date d_i values. All of the instances were randomly generated as follows: For each job i ($i = 1, \dots, n$), an integer processing time p_i and weight w_i were generated from the uniform distribution $[1, 100]$ and $[1, 10]$ respectively. Instance classes of varying hardness were generated by using different uniform distributions for generating the due dates d_i . The *due date range* factors are set to $R = \{0.2, 0.4, 0.6, 0.8, 1\}$ and the *due date tightness* factors are set to $Tf = \{0.2, 0.4, 0.6, 0.8, 1.0\}$, an integer due date d_i was randomly generated from the uniform distribution interval $[P * (1 - Tf - R/2), P * (1 - Tf + R/2)]$, where $P = \sum_{i=1 \dots n} p_i$. Five instances were generated for each of the 25 pairs of (R, Tf) values, yielding 125 instances for each instance of size n . The values of Tf and R provide characterizations of the instances. For instance, values of Tf close to 1 indicate that the due dates are *tight* and values close to zero indicate that the due dates are *loose*. High values of R indicate a *wide* due date range whereas small values indicate a *narrow* due date range.

Our new data test instances for TWT-SMSP-PM have additional maintenance characteristics in addition to job characteristics in the original data. These maintenance additions are explained as follows. For each data test instance, a total of 8 block-durations (T), and a total 5 different maintenance durations (ML) are considered. The original characteristics and the new maintenance setting would lead to a total of 20,000 data instances used to report our computational experience. Moreover, the values for T and ML are generated in relation to the processing time of jobs to have different characteristics similar to that used in closely related literature. For instance, Chen [5] generated instances to minimize the weighted total completion time in relation to the upper value (b) of the processing time interval $\{p_i \in [a, b]\}$ and set $T = \theta \times b$ with $\theta = 1, 1.5$, and 2 and $ml = \mu \times b$ with $\mu = 0.3$ and 0.6. On the other hand, Raza *et al.* [20] generated instances to minimize the total number of early and tardy jobs for which they are set $\theta = 1.5, 2, 2.5$, & 3 with $\mu = 0.3, 0.33, 1$, & 1.1.

For our data set of instances, we have generated different block processing and maintenance repair lengths to investigate wider options than those previously used in the literature. For each of the 125 instances, we have generated forty pair combinations of (θ, μ) with the following settings: eight values for θ with $\theta = 1, 1.5, 2, 2.5, 3, 4, 5$, & 6 and five values for μ with $\mu = 0.3, 0.4, 0.6, 0.8$, & 1.

Table2 summarizes the various characteristics of the data instances. The test instances can be characterized with respect to θ and μ values; small values of θ indicate shorter processing time interval T (one week) where only few jobs can be processed in a batch, whereas medium to large values of θ indicate larger processing time interval (one month, or

one quarter) where more jobs can be processed. Small values of μ indicate shorter repair length (one day) whereas larger μ values indicate larger repair time (2 days or more).

Table 2 Characteristics of New Data Test Instances

	Parameters	Variations	Experimented Values
<i>Jobs characteristics</i>	Number of jobs	4	30, 40, 50, 100
	Processing time variability	1	[1, 100]
	Weight variability	1	[1, 10]
	Due date range, R	5	0.2, 0.4, 0.6, 0.8, 1.0
<i>Maintenance characteristics</i>	Tardiness factor, TF	5	0.2, 0.4, 0.6, 0.8, 1.0
	Block Duration time, T	8	100, 150, 200, 250, 300, 400, 500, 600
	Maintenance time, ML	5	30, 40, 60, 80, 100
Total test instances		20000	

4.2 Average relative percentage deviation calculation

The effectiveness of an algorithm was measured in terms of the objective function value or the relative percentage deviation (RPD) from the best solution found by all algorithms (including the optimal solutions whenever found). Let H and B be a heuristic solution value and the best-known solution value for a given instance respectively. Then, RPD is computed as $RPD = \left\{ \frac{H-B}{B} \right\} * 100$. Unfortunately, this calculation of RPD often led to a division by zero, since for some instances a solution with zero total weighted tardiness was obtained. Therefore, throughout the computational analysis it has been assumed that for any set of instances RPD is calculated assuming that H represents the sum of heuristic values for the target set and B represents the sum of the best-known solution values for the set. For example, for $n=30$, the target set is the set of all instances of $n=30$.

4.3 Impact of the pushback strategy on the performance of dispatching rules

In this section, we demonstrate the high impact of the pushback strategy on improving the solution quality of all dispatching rule heuristics. Table 3 reports the averages of the objective values (OV) overall instances for each size of n for all the seven dispatching rules (R1 to R7) listed in Table 1. We have reported the results for the DR heuristics with and without the pushback strategy. The RPD values in Table 3 report the relative percentage deviation of the values of the compared algorithms. The results for ATC are obtained with a look-ahead parameter α computed using $\alpha = 5.5 - \tau - \rho$ where $\tau = \sum_i d_i / nP$ and $\rho = (d_{\max} - d_{\min}) / P$ as recommended in Pinedo [18].

From the results in Table 3, a number of remarks can be made. *First*, the performance of each dispatching rule is clearly improved by implementing the pushback strategy with no overhead in CPU time as shown in the last column CPU. On the contrary, CPU time is improved since job ranking that was performed in every iteration whereas it is done only on feasible jobs in the pushback strategy. The relative percentage improvements of the results of the “without pushback” rule from those “with pushback” reported under the column (%Imp) of Table 3 show the magnitude of impact of this simple rule. The maximum improvements were observed with values of 92, 94, 95 and 96% for $n=30, 40, 50$ and 100 respectively for the WMSPT rule, whereas the corresponding values for WMDD are 1% less and are much more variable for the ATC rule. Such good improvements may be due to the flexible structures of the generated solutions by the WMPST, ATC, and WMDD that allowed free spaces towards the end of blocks unlike the other more greedy dispatching rules based on WSPT per say. *Second*, the new proposed WMSPT rule performed best among all the seven DR rules on the average in both strategies. It therefore seems that the WMSPT rule is able to balance better the

penalty trade-offs between the lateness of a job and its processing time than its counterpart rules such as WMDD and ATC. *Third*, it is supervising to notice the huge improvement of WMSPT over its ancestors WSPT and WMS. WMSPT rule seems to be the best of both rules by alternating them over time. It should be also noted without the push back strategy the new dispatching rule was ranked first on $n=30$ and 40 and second on $n=50$ and 100 with insignificant differences. But the later requires a number of parameter settings and higher computational efforts. A final point to mention is that we have used Cplex with a cut-off time to solve the proposed mixed integer formulation of the problem, however, only few cases with special characteristics were solved to optimality, with a huge overall average per set worse than any of the above mentioned dispatching rules. Hence its corresponding results are omitted.

Table 3 Comparison of Dispatching Rules with and without pushback strategy

n	DRH	With Pushback			Without Pushback			%Imp	Best	CPlex	
		OV	RPD	CPU	OV	RPD	CPU		OV	OV	RPD
30	WEDD	37750	31.13	0.09	54627	89.76	0.09	65.32	28788	38971	35.37
	WMS	52615	82.77	0.21	79187	175.07	0.29	52.72			
	WSPT	36681	27.42	0.09	41803	45.21	0.09	39.35			
	WRA	64433	123.82	0.19	97341	238.13	0.24	48.00			
	ATC	30715	6.70	0.29	36404	26.45	0.38	74.69			
	WMDD	29405	2.14	0.20	36144	25.55	0.26	91.61			
	WMSPT	29334	1.90	0.20	35895	24.69	0.28	92.32			
40	WEDD	91958	43.76	0.11	128599	101.04	0.12	56.69	63967	85071	32.99
	WMS	111630	74.51	0.29	163715	155.94	0.42	52.22			
	WSPT	77751	21.55	0.11	88760	38.76	0.11	44.40			
	WRA	141926	121.87	0.27	206714	223.16	0.37	45.39			
	ATC	66476	3.92	0.42	78813	23.21	0.58	83.10			
	WMDD	64802	1.31	0.29	78316	22.43	0.39	94.18			
	WMSPT	64782	1.27	0.29	78207	22.26	0.38	94.28			
50	WEDD	135718	47.90	0.29	190859	107.99	0.30	55.64	91763	157562	71.71
	WMS	166364	81.30	0.98	248044	170.31	1.50	52.26			
	WSPT	113045	23.19	0.29	130153	41.84	0.30	44.57			
	WRA	214920	134.21	0.87	323809	252.87	1.25	46.93			
	ATC	94752	3.26	1.43	114338	24.60	2.09	86.76			
	WMDD	92976	1.32	0.97	114432	24.70	1.41	94.65			
	WMSPT	92864	1.20	0.97	114484	24.76	1.42	95.15			
100	WEDD	398333	35.97	0.29	593792	102.70	0.30	64.97	292947	NA	NA
	WMS	556962	90.12	0.98	862395	194.39	1.50	53.64			
	WSPT	372302	27.09	0.29	441542	50.72	0.30	46.60			
	WRA	723217	146.88	0.87	1117936	281.62	1.25	47.85			
	ATC	298449	1.88	1.43	380673	29.95	2.09	93.73			
	WMDD	296897	1.35	0.97	382371	30.53	1.41	95.58			
	WMSPT	295844	0.99	0.97	381144	30.11	1.42	96.72			

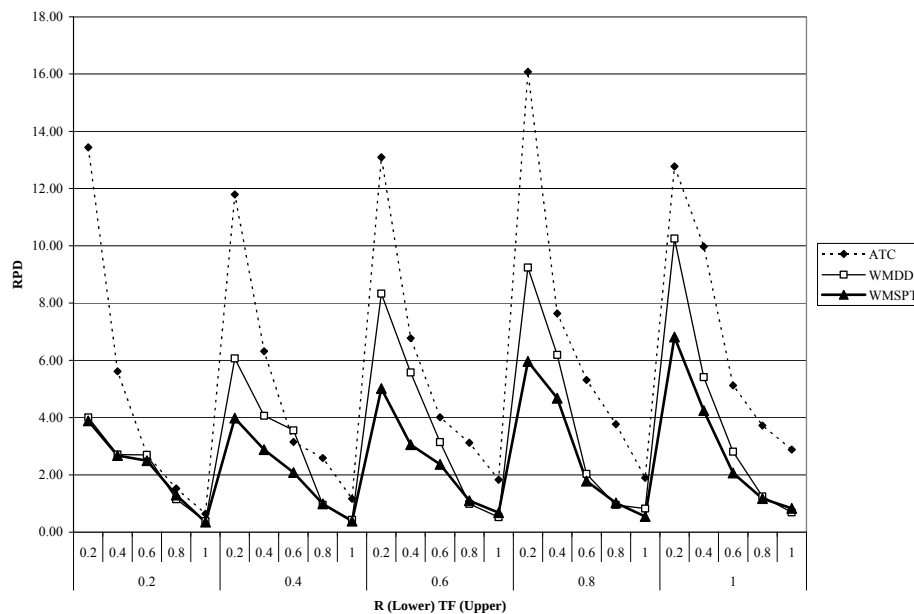
In our further analysis we will only consider the best performing DR heuristics with pushback strategy namely WMSPT, WMDD, and ATC. Table 4 presents a summary of the average improvement of the new proposed method. It can be seen that WMSPT is the best performing algorithm in terms of solution quality at a comparable CPU time to WMDD and much faster than ATC.

Table 4 Comparison of Cplex and the best performing heuristics with pushback strategy

		Pushback							Cplex
		WEDD	WMS	WSPT	WRA	ATC	WMDD	WMSPT	
RPD	30	31.13	82.76	27.42	123.82	6.69	2.14	1.90	35.37
	40	43.76	74.51	21.55	121.88	3.92	1.31	1.27	32.99
	50	47.90	81.30	23.19	134.21	3.26	1.32	1.20	71.71
	100	35.97	90.12	27.09	146.88	1.88	1.35	0.99	NA
CPU	30	0.09	0.21	0.09	0.19	0.29	0.20	0.20	15000
	40	0.11	0.29	0.11	0.27	0.42	0.29	0.29	20000
	50	0.29	0.98	0.29	0.87	1.43	0.97	0.97	25000
	100	0.29	0.98	0.29	0.87	1.43	0.97	0.97	NA

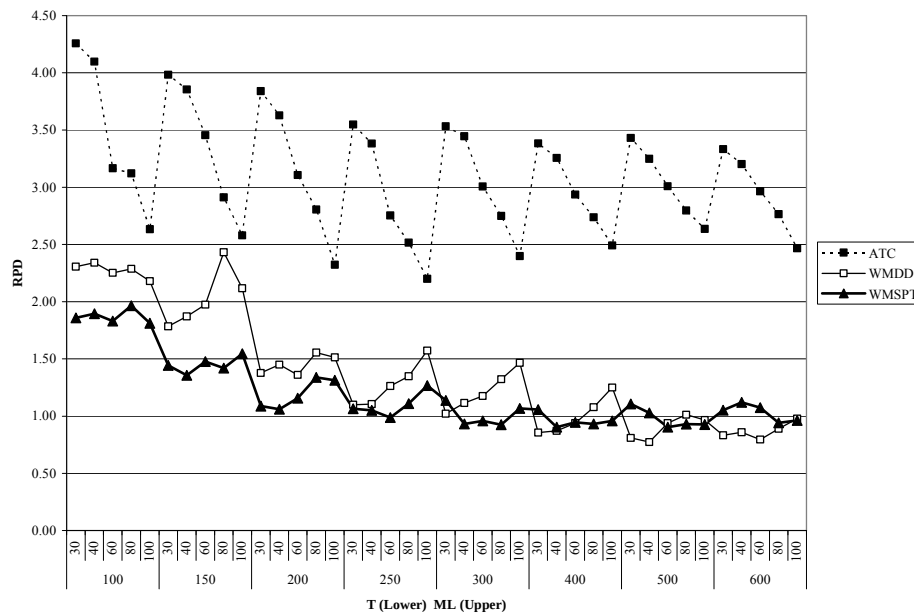
Further analyses were conducted in Figure 1 with respect to job characteristics and in Figure 2 for machine characteristics. The results in Figure 1 clearly show that when both the due date tightness (TF) factor is narrow (small values) the performance of DR heuristics are weak and they improve as TF becomes large value close to 1. In addition, as the due date range change values from small to large, the same performance pattern is repeated but with much worse performance. However, despite all TF and R variations, the ranking order of the DR heuristics is kept the same i.e. WMSPT first, WMDD second, and ATC third.

Figure 1: Heuristics Comparison by R and TF for all n



In Figure 2, we investigated the impact of changing the length or maintenance (ML) and working period (T) of the machine. It can be seen that as T increases, the DRH performance improves by having a lower deviation value. It is interesting to note that as T gets very large (over 400 units), the WMSPT and WMDD performance gap gets very close. However, the difference remains significant in favour of WMSPT rule at 0.95% confidence interval using T-test statistics.

Figure 2L: Heuristics Comparison by T and ML for all n



5 Conclusion

The most important finding in this paper is that WMSPT is a very competitive rule to the well-known classical dispatching rules in the literature. The pushback strategy was found to improve all the dispatching rules including the WMSPT heuristic which was to be the most effective and efficient strategy among all tested heuristics for the TWT-SMSP-PM with an equal computational complexity. Another advantage of the WMSPT rule is that it is a very simple to implement, parameter free dispatching rule when compared to the most competitive ATC rule which requires the parameter α as estimate. The current research is continuing to determine the impact of the problem's characteristics on the new dispatching rules and whether the performance of the new proposed dispatching rule can be extended to the periodic maintenance problems with different objectives.

Acknowledgements The authors would like to acknowledge the American University of Beirut Research Board to support the funding of this project.

References

1. Adiri, I., Bruno, J., Frosting, J. & Rinnoy, Kan, A.H.G., Single Machine flow-time scheduling with a single breakdown, *ACTA informatica*, 26, 679-696 (1989)
2. Biskup, D., & Feldmann M., Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates, *Computers & Operations Research*, 28, 787-801 (1989)

3. Cassady, C.R., & Kutanoglu, E., Integrating preventive maintenance planning and production scheduling for a single machine. *IEEE Transactions on reliability*. 54 (2), 304-309 (2005)
4. Chen, W-J., Minimizing total flow time in the single-machine scheduling problem with periodic maintenance, *Journal of Operational Research Society*, 57, 410-415 (2006)
5. Chen, W-J., An efficient algorithm for scheduling jobs on a machine with periodic maintenance, *Int J Adv Manuf Technol* 34 (1) 173-1182 (2007)
6. Chen, W-J., Minimizing number of tardy jobs on a single machine subject to periodic maintenance. *Omega* doi: 10.1016/j.omega.2008.01.001 (2008)
7. Crauwels, H.A.J., Potts, C.N., Van Wassenhove, L.N., Local search heuristics for the single machine total weighted tardiness scheduling problem, *INFORMS Journal on Computing*, 10 (3), 341-350 (1998)
8. Davenport, T.H. Competing on analytics. *Harvard Business Review*, 1-10, January (2006).
9. Emmons, H., One-Machine Sequencing to Minimize Certain Functions of Job Tardiness. *Operations Research* 17, 701-715 (1969)
10. Graves and Lee, Scheduling maintenance and semi-resumable jobs on a single machine. *Naval Research Logistics*. v46. 845-863, (1999)
11. Ji, M., He, Y., Cheng, T.C.E., Single-machine scheduling with periodic maintenance to minimize makespan. *Computers and Operations Research*, 34, 1764-1770 (2007)
12. Kanet, J.J. & Li, X., A weighted modified due date rule for sequencing to minimize weighted tardiness. *Journal of Scheduling* 7, 261-276 (2004)
13. Lee, C.Y., Machine scheduling with an availability constraint. *J. Global Optim.* 9, 395-416 (1996)
14. Lenstra, J.K., Rinnoy Kan, A.H.G. & Brucker, P., Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 4, 121-140 (1977)
15. Leon, V.J., & Wu, S.D., On scheduling with ready-times, due-dates and vacations. *Naval Research Logistics*, 39, 53-65 (1992)
16. Liao, C.J., & Chen, WJ., Single machine scheduling with periodic maintenance and non-resumable jobs. *Computers & Operations Research*, 30, 1335-1347 (2003)
17. Morton, T. E., & Pentico, D., *Heuristic Scheduling Systems*. John Wiley & Sons, Inc., New York (1993)
18. Pinedo, M., *Scheduling: Theory, Algorithms and Systems*. 2nd edition, Prentice Hall (2002)
19. Qi, X, Chen T. & Tu, F., Scheduling the maintenance on a single machine. *Journal of the Operational Research Society*, 50, 1071-1078 (1999)
20. Raza, S.A., & Al-Turk, U.M., A comparative study of heuristic algorithms to solve maintenance scheduling problem. *Journal of Quality in Maintenance Engineering*, 13(4), 398-410 (2007)
21. Smith, W.E., Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3, 59-66 (1956)
22. Sortrakul, N., & Cassad, C.R., Genetic Algorithms for the weighted expected tardiness integrated preventive maintenance scheduling for a single machine. *Journal of Quality in Maintenance Engineering*, 13 (1) 49-61 (2007)
23. A. Volgenant, E. Teerhuis, Improved heuristics for the n-job single-machine weighted tardiness problem. *Computers & OR* 26(1): 35-44 (1999)
24. Vopsalainon, A. P. J. & Morton, I. E., Priority rules for job shops with weighted tardiness costs. *Management Science*, 33 1035-1047 (1987)

An exact algorithm for the precedence-constrained single-machine scheduling problem

Shunji Tanaka · Shun Sato

Abstract This study proposes an exact algorithm for the precedence-constrained single-machine scheduling problem to minimize total job completion cost where machine idle time is forbidden. The proposed algorithm is an extension of the authors' previous algorithms for the problem without precedence-constraints and is based on the SSDP (Successive Sublimation Dynamic Programming) method. Numerical experiments will show that it can solve all instances with up to 50 jobs of the precedence-constrained total weighted tardiness and total weighted earliness-tardiness problems, and most instances with 100 jobs of the former problem.

1 Introduction

In this study we consider the single-machine scheduling problem to minimize total job completion cost subject to general precedence constraints ($1|prec|\sum f_j(t)$). A special class of this problem, the precedence-constrained single-machine problem to minimize total weighted completion time ($1|prec|\sum w_j C_j$) is well studied in the literature [1–7] and instances with up to 100 jobs were optimally solved even twenty years ago [3]. However, to the best of the authors' knowledge, there is no study on exact algorithms for the problem with more general objective functions. This motivates us to construct an exact algorithm for this class of problems.

For the single-machine problem without precedence constraints, the first author and his colleagues have already proposed efficient exact algorithms that can solve instances with up to 300 jobs when machine idle time is forbidden [8] and with up to 200 jobs when it is permitted [9, 10]. This study extends these algorithms to the precedence-constrained problem, but only the case when machine idle time is forbidden is considered. Numerical experiments will show that the proposed algorithm can solve all 50 jobs instances of the precedence-constrained total weighted tardiness problem ($1|prec|\sum w_i T_i$) and total weighted earliness-tardiness problem ($1|prec|\sum (\alpha_i E_i + \beta_i T_i)$), and most of 100 jobs instances of $1|prec|\sum w_i T_i$.

Shunji Tanaka and Shun Sato
Graduate School of Electrical Engineering, Kyoto University
E-mail: tanaka@kuee.kyoto-u.ac.jp

2 Problem description and network representation

Consider that n jobs (job 1, job 2, ..., job n) are to be processed on a machine that can process at most one job at a time. Each job $i \in \mathcal{N} = \{1, 2, \dots, n\}$ is given an integer processing time $p_i > 0$. An integer cost function $f_i(t)$ ($t \geq p_i$) is also given for each job i and the cost $f_i(C_i)$ is incurred when job i is completed at C_i . On some pairs of jobs i and j ($i, j \in \mathcal{N}, i \neq j$), the precedence constraint that job i should precede job j ($i \rightarrow j$) is imposed. All the precedence constraints are specified by an acyclic directed graph $\overline{G_P} = (V_P, \overline{A_P})$ with $V_P = \{1, 2, \dots, n\}$ such that $i \rightarrow j$ when $(i, j) \in \overline{A_P}$. $\overline{A_P}$ is assumed to be transitively closed: if $(i, k), (k, j) \in \overline{A_P}$, then $(i, j) \in \overline{A_P}$. The minimal expression of the precedence constraints is given by $G_P = (V_P, A_P)$, where $A_P \subseteq \overline{A_P}$ is the transitive reduction of $\overline{A_P}$. Preemption and machine idle time are forbidden and hence all the jobs should be processed in the interval $[0, T]$ where $T = \sum_{i \in \mathcal{N}} p_i$. The objective is to find a job sequence that satisfies the precedence constraints and that minimizes total job completion cost $\sum_{i \in \mathcal{N}} f_i(C_i)$.

We treat this problem as a constrained shortest path problem on a directed graph. Let us introduce a dummy job $n+1$ with $p_{n+1} = 1$, $f_{n+1}(t) \equiv 0$ that should be completed twice at time 0 and $T+1$. Next, define a node set V by

$$V = \{v_{n+1,0}\} \cup V_O \cup \{v_{n+1,T+1}\}, \quad (1)$$

$$V_O = \{v_{it} \mid i \in \mathcal{N}, p_i \leq t \leq T\}. \quad (2)$$

An arc set A is defined by

$$A = \{(v_{i,t-p_j}, v_{jt}) \mid v_{i,t-p_j}, v_{jt} \in V\}, \quad (3)$$

where the length of an arc $(v_{i,t-p_j}, v_{jt}) \in A$ is given by $f_j(t)$. Then, our problem is equivalent to find a shortest path from the source node $v_{n+1,0}$ to the sink node $v_{n+1,T+1}$ such that v_{is} is visited exactly once for any $i \in \mathcal{N}$ and that v_{is} and v_{jt} in the path satisfy $s \leq t - p_j$ if $(i, j) \in A_P$. The optimal objective value is identical to the shortest path length, and v_{it} visited in the shortest path corresponds to the completion of job i at t in an optimal solution.

Hereafter, we use the following notation and definitions. A set of nodes visited in a path from $v_{n+1,0}$ to $v_{n+1,T+1}$ is denoted by $\mathcal{P}$, and the path corresponding to $\mathcal{P}$ is referred to as “path $\mathcal{P}$ ” for simplicity. Let $L(\mathcal{P})$ be the length of a path $\mathcal{P}$. Namely, $L(\mathcal{P})$ is defined by

$$L(\mathcal{P}) = \sum_{\substack{v_{it} \in \mathcal{P} \\ i \in \mathcal{N}}} f_i(t). \quad (4)$$

The number of occurrences of v_{is} in $\mathcal{P}$ is denoted by $\mathcal{V}_i(\mathcal{P})$, which is defined by

$$\mathcal{V}_i(\mathcal{P}) = |\{v_{it} \mid v_{it} \in \mathcal{P}\}|. \quad (5)$$

Then, the constraints that $\mathcal{P}$ should visit v_{is} ($i \in \mathcal{N}$) exactly once can be written by

$$\mathcal{V}_i(\mathcal{P}) = 1, \quad i \in \mathcal{N}. \quad (6)$$

In addition, the precedence constraints can be expressed by

$$s \leq t - p_j, \quad (i, j) \in A_P, v_{is}, v_{jt} \in \mathcal{P}, \quad (7)$$

or, equivalently,

$$\sum_{v_{is} \in \mathcal{P}} s \leq \sum_{v_{jt} \in \mathcal{P}} t - p_j, \quad (i, j) \in A_P. \quad (8)$$

We denote the set of all the feasible paths satisfying these constraints by $\mathcal{Q}$. Then, our problem (P) can be formulated as follows.

$$(P) : \min_{\mathcal{P}} L(\mathcal{P}) \quad \text{s.t. } \mathcal{P} \in \mathcal{Q}. \quad (9)$$

3 Lagrangian relaxation and dynamic programming

The proposed algorithm is based on the SSDP (Successive Sublimation Dynamic Programming) method [11, 12] as our previous algorithms [8–10]. This algorithm is similar to the cutting plane algorithm: it first computes a lower bound by a simple relaxation of the original problem, and then improves it successively by adding constraints (cuts) to the relaxation until the gap between lower and upper bounds becomes zero. Since all the relaxations are solved by dynamic programming, it is inevitable that the number of dynamic programming states increases as the number of added constraints increases. To cope with it, unnecessary states are eliminated in the course of the algorithm, which is the key point of the SSDP method.

This section gives the relaxations used in the proposed algorithm. They do not differ much from those in our previous algorithm [8]. The primary difference is that all or part of the precedence constraints are penalized by Lagrangian relaxation. However, it does not follow that the original dynamic programming is applicable to the relaxations considered here. One of our main contributions is that it is applicable by simple modifications without increasing the time complexity, which will be explained in 3.4.

3.1 Lagrangian relaxation

We start from a Lagrangian relaxation of the original problem (P). Let us penalize the constraints on the number of occurrences (6) by μ_i^O ($i \in \mathcal{N}$) and the precedence constraints of the form (8) by $\mu_{ij}^P \geq 0$ ($(i, j) \in A_P$). Then, we obtain the following relaxation (LR₀):

$$(LR_0) : \min_{\mathcal{P}} \left\{ L(\mathcal{P}) + \sum_{i \in \mathcal{N}} \mu_i^O (1 - \mathcal{V}_i(\mathcal{P})) + \sum_{(i,j) \in A_P} \mu_{ij}^P \left(\sum_{v_{is} \in \mathcal{P}} s - \sum_{v_{jt} \in \mathcal{P}} t + p_j \right) \right\}. \quad (10)$$

By introducing ω_i ($i \in \mathcal{N}$) defined by

$$\omega_i = \sum_{(j,i) \in A_P} \mu_{ji}^P - \sum_{(i,j) \in A_P} \mu_{ij}^P, \quad (11)$$

(10) can be rewritten as

$$(LR_0) : \min_{\mathcal{P}} L_R(\mathcal{P}; \boldsymbol{\mu}^O, \boldsymbol{\mu}^P) + \sum_{i \in \mathcal{N}} \mu_i^O + \sum_{(i,j) \in A_P} \mu_{ij}^P p_j, \quad (12)$$

where $L_R(\mathcal{P}; \boldsymbol{\mu}^O, \boldsymbol{\mu}^P)$ is defined by

$$\begin{aligned} L_R(\mathcal{P}; \boldsymbol{\mu}^O, \boldsymbol{\mu}^P) &= L(\mathcal{P}) - \sum_{i \in \mathcal{N}} \mu_i^O \mathcal{V}_i(\mathcal{P}) + \sum_{(i,j) \in A_P} \mu_{ij}^P \left(\sum_{v_{is} \in \mathcal{P}} s - \sum_{v_{jt} \in \mathcal{P}} t \right) \\ &= \sum_{\substack{v_{it} \in \mathcal{P} \\ i \in \mathcal{N}}} f_i(t) - \sum_{\substack{v_{it} \in \mathcal{P} \\ i \in \mathcal{N}}} \mu_i^O - \sum_{\substack{v_{it} \in \mathcal{P} \\ i \in \mathcal{N}}} \omega_i t \\ &= \sum_{\substack{v_{it} \in \mathcal{P} \\ i \in \mathcal{N}}} (f_i(t) - \mu_i^O - \omega_i t). \end{aligned} \quad (13)$$

From (13), we can see that (LR₀) for a set of multipliers $\boldsymbol{\mu}^O$ and $\boldsymbol{\mu}^P$ is equivalent to the unconstrained shortest path problem on G , where the length of an arc $(v_{i,t-p_j}, v_{jt}) \in A$ is given not by $f_j(t)$, but by $f'_j(t) = f_j(t) - \mu_j^O - \omega_j t$. Therefore, (LR₀) can be solved by dynamic programming in $O(nT)$ time, as the relaxation of the problem without precedence constraints [13].

3.2 Improvement by the constraints on successive jobs

To improve the lower bound obtained by (LR_0) , the following constraints are imposed on (LR_0) as in the previous algorithms [12, 8–10]:

For any $i \in \mathcal{N}$, nodes corresponding to job i , i.e., v_{i*} should not be visited more than once in any $\lambda + 1 > 0$ successive nodes in a path.

A subset of $\mathcal{Q}$ satisfying these constraints on successive nodes is denoted by $\mathcal{Q}_\lambda$, and the relaxation with the constraints is denoted by (LR_λ) . Here, we only consider (LR_1) and (LR_2) . By defining

$$A_S = A \setminus \{(v_{i,t-p_i}, v_{it}) \mid v_{i,t-p_i}, v_{it} \in V_O\}, \quad (14)$$

we introduce a reduced graph $G_S = (V, A_S)$. Then, (LR_1) becomes equivalent to the unconstrained shortest path problem on G_S . On the other hand, (LR_2) remains the constrained shortest path problem even on G_S . The time complexities of (LR_1) and (LR_2) are $O(nT)$ and $O(n^2T)$, respectively.

3.3 Improvement by the constraints on adjacent pairs of jobs

The second constraints are derived from the dominance theorem of dynamic programming [14] for adjacent pairs of jobs [15, 8–10]. For the problem without precedence constraints, they restrict processing orders of adjacent pairs of jobs by checking their total costs. Here, we slightly modify them to ensure the consistency with the precedence constraints.

Consider that two jobs i and j ($i, j \in \mathcal{N}$, $i \neq j$) are processed adjacently. If $(i, j) \in \bar{A}_P$ and $(i, j) \notin A_P$ (or, $(j, i) \in \bar{A}_P$ and $(j, i) \notin A_P$), this situation cannot happen because there should exist another job k satisfying $(i, k), (k, j) \in \bar{A}_P$ (or, $(j, k), (k, i) \in \bar{A}_P$). In this case, we forbid the two jobs i and j to be processed adjacently. On the other hand, if $(i, j) \in A_P$ or $(j, i) \in A_P$, the two jobs can be processed adjacently, but their processing order is restricted by the precedence constraint. If no precedence constraint exists between the jobs i and j , i.e., $(i, j), (j, i) \notin \bar{A}_P$, we compare the total completion costs of the two jobs when they are completed at t : $f_i(t - p_j) + f_j(t)$ when job i precedes job j , and $f_j(t - p_i) + f_i(t)$ when job j precedes job i . Then, the processing order that yields the larger cost is forbidden. If the two costs are identical, either (but not arbitrary) processing order can be forbidden without loss of optimality. To summarize, the processing orders of any adjacent pairs of jobs can be restricted and these restrictions are imposed on (LR_2) as constraints. This yields a new relaxation $(\widehat{LR}_2)$.

In the network representation, these adjacency constraints eliminate from G_S , those edges corresponding to the forbidden processing orders. Thus, we define $\widehat{G}_S = (V, \widehat{A}_S)$ by

$$\widehat{A}_S = A_S \setminus \{(v_{i,t-p_j}, v_{jt}) \mid \text{job } j \text{ cannot be preceded adjacently by job } i \text{ at } t\}. \quad (15)$$

Then, $(\widehat{LR}_2)$ becomes equivalent to the shortest path problem on $\widehat{G}_S$ under the constraints on three successive nodes. The time complexity of $(\widehat{LR}_2)$ does not increase from (LR_2) and it can be solved in $O(n^2T)$ time. Let $\widehat{\mathcal{Q}}_2$ denote a subset of $\mathcal{Q}_2$ composed of feasible paths for $(\widehat{LR}_2)$.

3.4 Recovering the relaxed constraints

In our previous algorithms [8–10], the relaxed constraints (6) are partially recovered to generate a better relaxation. More specifically, for some $\mathcal{M} = \{\phi_1, \dots, \phi_m\} \subseteq \mathcal{N}$,

$$\mathcal{V}_{\phi_k}(\mathcal{P}) = 1, \quad 1 \leq k \leq m, \quad (16)$$

are imposed on the relaxation corresponding to $(\widehat{\text{LR}}_2)$. It is easy to see that the relaxation approaches to the original problem as $m = |\mathcal{M}|$ increases and they become equivalent when $m = n$ (to be more precise, when $m = n - 1$ if machine idle time is forbidden). This relaxation is solvable in $O(n^2 2^m T)$ time.

In this study some of the precedence constraints (7), i.e.,

$$s \leq t - p_i, \quad (\phi_k, i) \in A_P, \quad v_{\phi_k s}, v_{it} \in \mathcal{P}, \quad 1 \leq k \leq m, \quad (17)$$

$$s \leq t - p_{\phi_k}, \quad (i, \phi_k) \in A_P, \quad v_{is}, v_{\phi_k t} \in \mathcal{P}, \quad 1 \leq k \leq m, \quad (18)$$

are recovered at the same time. In addition, μ_{ij}^P is set to zero if $i, j \in \mathcal{M}$. Clearly, the relaxation generated in this manner, which is referred to as $(\widehat{\text{LR}}_2^m)$, becomes equivalent to (P) when $\mathcal{M} = \mathcal{N}$. Therefore, an optimal solution of (P) is obtainable by solving $(\widehat{\text{LR}}_2^m)$ with $\mathcal{M}$ increased until the gap between the current upper bound and the optimal objective value of $(\widehat{\text{LR}}_2^m)$ becomes less than one (note that the optimal objective value of (P) is integral).

Our claim here is that $(\widehat{\text{LR}}_2^m)$ is solvable in $O(n^2 2^m T)$ time and hence the time complexity does not increase even under the existence of the precedence constraints (17) and (18). First, assume that only the precedence constraints (16) are imposed on $(\widehat{\text{LR}}_2)$ and denote the resulting relaxation by $(\widehat{\text{LR}}_2^m)$. To solve it, define an m -dimensional vector $\mathbf{q}_i^m$ ($i \in \mathcal{N}$) by $\mathbf{q}_i^m = (q_{i1}, \dots, q_{im})$, where

$$q_{ik} = \begin{cases} 1 & i = \phi_k, \\ 0 & \text{otherwise.} \end{cases} \quad (19)$$

Let us also define an m -dimensional vector $\mathbf{q}_{n+1}^m$ by $\mathbf{q}_{n+1}^m = (0, \dots, 0)$. Next, we introduce a directed graph $G_S^m = (V^m, A_S^m)$. The node set V^m is defined by

$$V^m = \{v_{n+1,0}^{\mathbf{0}_m}\} \cup V_O^m \cup \{v_{n+1,T+1}^{\mathbf{1}_m}\}, \quad (20)$$

$$V_O^m = \{v_{it}^{\mathbf{b}} \mid v_{it} \in V_O, \mathbf{q}_i^m \leq \mathbf{b} \leq \mathbf{1}_m\}, \quad (21)$$

where $\mathbf{0}_m$ and $\mathbf{1}_m$ denote m -dimensional vectors whose elements are all zero and all one, respectively. The arc set A_S^m is defined by

$$A_S^m = \{(v_{i,t-p_j}^{\mathbf{b}-\mathbf{q}_j^m}, v_{jt}^{\mathbf{b}}) \mid (v_{i,t-p_j}, v_{jt}) \in \widehat{A}_S, \mathbf{q}_i^m + \mathbf{q}_j^m \leq \mathbf{b} \leq \mathbf{1}_m\}, \quad (22)$$

and the length of an arc $(*, v_{jt}^{\mathbf{b}})$ is given by $f_j'(t)$. Then, the shortest path problem on G_S^m under the constraints on three successive nodes is equivalent to $(\widehat{\text{LR}}_2^m)$. To solve it by (forward) dynamic programming, the constrained shortest path is recursively computed from the source node $v_{n+1,0}^{\mathbf{0}_m}$ to $v_{jt}^{\mathbf{b}}$ through $(v_{i,t-p_j}^{\mathbf{b}-\mathbf{q}_j^m}, v_{jt}^{\mathbf{b}})$. Since this shortest path is such that the number of occurrences of $v_{\phi_k}^*$ ($1 \leq k \leq m$) is b_k , i.e., the k th element of $\mathbf{b}$, it is ensured that $v_{\phi_k}^*$ appears exactly once for all k ($1 \leq k \leq m$) in the obtained shortest path from $v_{n+1,0}^{\mathbf{0}_m}$ to $v_{n+1,T+1}^{\mathbf{1}_m}$.

Next, consider $(\widehat{LR}_2^m)$, i.e., $(\widetilde{LR}_2^m)$ with the precedence constraints (17) and (18). As we have already seen, the number of occurrences of $v_{\phi_k^*}^*$ in the path from $v_{n+1,0}^0$ to v_{jt}^b is equal to b_k . It implies that the path breaks the precedence constraint (17) if $(\phi_k, j) \in A_P$ and $b_k = 0$. Similarly, it breaks the constraint (18) if $(j, \phi_k) \in A_P$ and $b_k = 1$. Therefore, we can satisfy the precedence constraints (17) and (18) by removing all such nodes from G_S^m . Clearly, the time complexity of the constrained shortest path problem on G_S^m does not change even when some nodes are removed. Hence, we can conclude that $(\widehat{LR}_2^m)$ is solvable in $O(n^2 2^m T)$ time.

Notice that $i \rightarrow j$ may not be satisfied when $i \notin \mathcal{M}$, $j \in \mathcal{M}$, or, $i \in \mathcal{M}$, $j \notin \mathcal{M}$ because the graph constructed in the above manner only ensures that $i \rightarrow j$ is satisfied if both the corresponding nodes appear in the path from $v_{n+1,0}^0$ to $v_{n+1,T+1}^1$. However, $i \rightarrow j$ for any $(i, j) \in A_P$ is satisfied when $\mathcal{M} = \mathcal{N}$ because all the jobs should appear exactly once because the constraints (6) are all recovered.

We can apply the above arguments not only to the minimal precedence constraints ($i \rightarrow j$ for $(i, j) \in A_P$), but also to the transitive closure ($i \rightarrow j$ for $(i, j) \in \overline{A_P}$), which improves the lower bound and reduces the graph more. To summarize, we are to solve the shortest path problem on a directed graph $\widehat{G}_S^m = (\widehat{V}^m, \widehat{A}_S^m)$ under the constraints on three successive nodes, where

$$\widehat{V}^m = V^m \setminus \left(\{v_{it}^b \mid \exists k, (\phi_k, i) \in \overline{A_P}, b_k = 0\} \cup \{v_{it}^b \mid \exists k, (i, \phi_k) \in \overline{A_P}, b_k = 1\} \right), \quad (23)$$

$$\widehat{A}_S^m = \{(v_{i,t-p_j}^{b-q_j^m}, v_{jt}^b) \mid (v_{i,t-p_j}^{b-q_j^m}, v_{jt}^b) \in A_S^m, v_{i,t-p_j}^{b-q_j^m}, v_{jt}^b \in \widehat{V}^m\}. \quad (24)$$

3.5 Improvement by slack variable optimization

Since the precedence constraints (7) are inequality constraints, the lower bound can be improved by converting them to equality constraints with slack variables and then solving optimization problems with regard to the slack variables [4]. We apply this method to all the relaxations (LR_1) , $(\widehat{LR}_2)$ and $(\widetilde{LR}_2^m)$.

4 Network reduction

As we can see from 3.4, the size of the graph in the network representation or, equivalently, the number of dynamic programming states to solve the relaxations increases as the number of added constraints increases. Therefore, we apply four methods to reduce the size of the graph [12, 8–10].

4.1 Reduction by upper bound

Unnecessary dynamic programming states can be eliminated by computing a lower bound passing through each dynamic programming states and comparing it with an upper bound [12]. In the network representation, we are to compute shortest paths passing through some node (in the case of (LR_1)) or arc (in the case of $(\widehat{LR}_2)$ and $(\widetilde{LR}_2^m)$) by concatenating two shortest paths that reach that node (arc) from the source node and from the sink node, respectively. Such paths are obtainable by applying dynamic programming in both forward and backward manners. For details, please refer to [12, 8].

4.2 Reduction by dominance theorem of dynamic programming

For $(\widehat{LR}_2^m)$, dominance of successive jobs is checked to eliminate arcs as in [10]. In the case of forward dynamic programming, we concentrate on the sequences of the last (up to six) nodes of all the feasible paths that reach v_{jt}^b from $v_{n+1,0}^{0m}$ through $(v_{i,t-p_j}^{b-q_j^m}, v_{jt}^b)$. Next, for each sequence of the nodes, we check whether interchanging their order improves the total cost of the corresponding partial schedule to show the existence of a dominating path. We assume that a dominating path exists even when only a partial schedule with the equal cost is found, but it is “better” under an appropriate tie-breaking rule. If there exists such a dominating path for every possible feasible path, the arc $(v_{i,t-p_j}^{b-q_j^m}, v_{jt}^b)$ can be eliminated. To break ties, the lexicographical order of job sequences is employed where the processing order of the best solution of (P), i.e., the solution yielding the best upper bound becomes the first in the order. This lexicographical order is also utilized as a tie-breaking rule in 3.3 to ensure the consistency.

4.3 Reduction by Constraint Propagation

The constraint propagation technique [16–19] is exploited by the job time windows computed from the current graph [10]. In the case of (LR_1) , for example, the time window $[\underline{r}_i, \bar{d}_i]$ of job i can be computed from G_S by

$$\underline{r}_i = \min_{v_{it} \in V_O} t - p_i, \quad \bar{d}_i = \max_{v_{it} \in V_O} t. \quad (25)$$

Then, their consistency is tested. If, as a result, the time window can be reduced to $[\underline{r}'_i, \bar{d}'_i]$, those nodes v_{it} such that $\underline{r}_i \leq t - p_i < \underline{r}'_i$, or $\bar{d}'_i < t \leq \bar{d}_i$ are eliminated. As the consistency tests, immediate selection [16, 18], edge-finding [16, 17, 19], not-first/not-last [16, 17, 19], and one more simple elimination rule are applied in the proposed algorithm.

4.3.1 Immediate selection

The precedence relation $i \rightarrow j$ can be induced from job time windows if

$$\bar{d}_i - \underline{r}_j < p_i + p_j. \quad (26)$$

We add such relations to the precedence graph $\overline{G}_P$ and utilize them for eliminating unnecessary nodes of $\widehat{G}_S^m$ (see (23) in 3.4). Conversely, job time windows can be updated by precedence relations. More specifically, $\underline{r}_i$ can be updated by

$$\underline{r}'_i = \max\{\underline{r}_i, \underline{C}(\mathcal{A})\}, \quad (27)$$

where $\mathcal{A} = \{j \mid (j, i) \in \overline{A}_P\}$, and $\underline{C}(\mathcal{A})$ denotes a lower bound of the maximum completion time of jobs in $\mathcal{A}$ calculated by

$$\underline{C}(\mathcal{A}) = \max_{j \in \mathcal{A}} (\underline{r}_j + \sum_{\substack{k \in \mathcal{A} \\ \underline{r}_k \geq \underline{r}_j}} p_k). \quad (28)$$

The dual argument holds for $\bar{d}_i$ and it can be updated by successor jobs.

4.3.2 Edge-finding

If job i and a set of jobs $\mathcal{A} \subseteq \mathcal{N} \setminus \{i\}$ satisfy

$$\max_{j \in \mathcal{A}} \bar{d}_j - \min_{j \in \mathcal{A} \cup \{i\}} \underline{r}_j < \sum_{j \in \mathcal{A} \cup \{i\}} p_j, \quad (29)$$

job i should be scheduled after all the jobs in $\mathcal{A}$ and $\underline{r}_i$ can be updated by

$$\underline{r}'_i = \max\{\underline{r}_i, \underline{C}(\mathcal{A})\}. \quad (30)$$

The dual argument holds for $\bar{d}_i$.

4.3.3 Not-first/not-last

If job i and a set of jobs $\mathcal{A} \subseteq \mathcal{N} \setminus \{i\}$ satisfy

$$\max_{j \in \mathcal{A}} \bar{d}_j - \underline{r}_i < \sum_{j \in \mathcal{A} \cup \{i\}} p_j, \quad (31)$$

at least one job in $\mathcal{A}$ should precede job i and $\underline{r}_i$ can be updated by

$$\underline{r}'_i = \max\left\{\underline{r}_i, \min_{j \in \mathcal{A}} (\underline{r}_j + p_j)\right\}. \quad (32)$$

The dual argument holds for $\bar{d}_i$.

4.3.4 Direct elimination

It is clear that C_i , i.e., the completion time of job i satisfies $C_i \leq \bar{d}_j - p_j$ if job i precedes job j , and $C_i \geq \underline{r}_j + p_i + p_j$ if job i is preceded by job j . Therefore, for any $(i, j) \notin \bar{A}_p$, the node v_{it} such that $\bar{d}_j - p_j + 1 \leq t \leq \underline{r}_j + p_i + p_j - 1$ are eliminated.

4.4 Node compression

To reduce the required memory space, up to three successive nodes are compressed into one super-node [10]. This node compression is applied to those nodes whose in-degrees are one.

5 Proposed algorithm

Our proposed algorithm is summarized as follow.

Stage 1 An initial upper bound UB is computed. Then, the conjugate subgradient algorithm [20] is applied to the Lagrangian dual corresponding to (LR₁). After it is terminated, the reduction techniques in 4.1 and 4.3 are applied. The algorithm is terminated without entering the next stage if the gap between the best lower bound and UB is less than one.

Stage 2 The multipliers are re-adjusted by the conjugate subgradient algorithm for the Lagrangian dual corresponding to (LR_2) . An upper bound is computed every 5 iterations and UB is updated if necessary. The reduction techniques in 4.1 and 4.3 are applied every time when the best lower bound or UB is updated. The algorithm is terminated without entering the next stage if the gap between the best lower bound and UB is less than one.

Stage 3 The relaxation $(\widehat{LR}_2^m)$ is solved with $\mathcal{M}$ increased from $\mathcal{M} = \emptyset$. To solve $(\widehat{LR}_2^m)$, the forward or backward dynamic programming is applied in turns and all the reduction techniques are applied. When $\mathcal{M}$ is increased, an upper bound is computed and UB is updated if necessary. Then, multipliers are re-adjusted by the conjugate subgradient algorithm for a smaller number of iterations than those in Stage 1 and Stage 2. After it is terminated, new jobs are chosen from $\mathcal{N} \setminus \mathcal{M}$ and added to $\mathcal{M}$. It is repeated until the gap between the lower bound and UB becomes less than one.

In this algorithm, an upper bound is computed in a similar way to those in [8–10]: a solution of a relaxation is converted to a feasible solution of (P), and then the enhanced dynasearch [21, 22] is applied to improve it. To convert to a feasible solution, we first remove all the jobs that break the precedence constraints or that occur more than once. Then, the jobs that do not appear in the partial schedule are inserted into it optimally or greedily. The enhanced dynasearch is extended to our problem simply by forbidding moves that break the precedence constraints. The initial upper bound is computed in the same manner, except that the length of the partial schedule is zero and all the jobs are inserted greedily.

In [8–10], the Lagrangian multipliers obtained in Stage 2 are also used in Stage 3, while in the proposed algorithm they are re-adjusted for $(\widehat{LR}_2^m)$ in Stage 3. It is because the duality gap of our problem is larger than that of the problem without precedence constraints, and hence it is necessary to improve the lower bound as much as possible. Another reason is that if jobs i and j appear in $\mathcal{M}$, μ_{ij}^p is set to zero to remove the corresponding penalty term from the objective function, which affects the optimal solution of the Lagrangian dual much.

In Stage 3, the jobs to be added to $\mathcal{M}$ are determined as follows. For each job $i \in \mathcal{M} \setminus \mathcal{N}$,

$$\sum_{(j,i) \in A_p} \mu_{ji}^p + \sum_{(i,j) \in A_p} \mu_{ij}^p \quad (33)$$

is computed and a job with a larger (33) is chosen first. Ties are broken by the occurrences of corresponding nodes in $\widehat{G}_S^m$, and a job that occurs less frequently is chosen first.

6 Numerical experiments

The proposed algorithm is applied to the precedence-constrained single-machine total weighted tardiness problem $(1|prec|\sum w_i T_i)$ and the precedence-constrained single-machine total weighted earliness-tardiness problem without machine idle time $(1|prec|\sum(\alpha_i E_i + \beta_i T_i))$. Instances are generated from the OR-Library instances of $1||\sum w_i T_i$ with 40, 50 and 100 jobs (available from <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html>). In the OR-Library instances, the processing time p_i , the tardiness weight w_i and the due date d_i of each job i were generated from the integer uniform distributions $[1, 100]$ and $[1, 10]$, $[T(1 - TF - RDD/2), T(1 - TF + RDD/2)]$, respectively. Here, TF and RDD are the tardiness factor and the range of due dates, respectively, which are chosen from $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. There

are five instances for each combination of n , TF and RDD, and hence 125 instances for each n . To generate instances of $1|prec|\sum w_i T_i$ from the OR-Library instances, precedence constraints are added as in [3,4]: for every pair of i and j ($i, j \in \mathcal{N}$, $i < j$), whether the precedence constraint $i \rightarrow j$ is imposed on or not is determined by the specified probability P . The instances of $1|prec|\sum(\alpha_i E_i + \beta_i T_i)$ are generated by adding precedence constraints in a similar way to the instances of $1||\sum(\alpha_i E_i + \beta_i T_i)$ used in [8], which were generated from the OR-Library instances by choosing the tardiness weight $\beta_i = w_i$ and by generating the earliness weight α_i from the uniform distribution $[1, 10]$. We choose P as $P = 0.005, 0.01, 0.02, 0.05, 0.1, 0.2$ and thus there are 125 instances for each combination of n , P and the problem type. The computation is performed on a 3.4GHz Pentium4 desktop computer with 1GB RAM. The maximum memory size for storing the network structure (dynamic programming states) is restricted to 768MB.

The results are shown in Tables 1 and 2, where average (ave) and maximum (max) computation times over optimally solved instances (opt) are shown in seconds. For comparison, the instances without precedence constraints ($P = 0.0$) are solved by an improved version of the algorithm in [8]. Results for the instances with 100 jobs of $1|prec|\sum(\alpha_i E_i + \beta_i T_i)$ are omitted because more than 30% instances could not be solved. From the tables, we can verify that the proposed algorithm successfully solved all the instances with 40 and 50 jobs, and most instances with 100 jobs of $1|prec|\sum w_i T_i$ after exhausting computation. Compared to the problem without precedence constraints, the duality gap in the precedence-constrained problem is larger and the problem is harder to solve. However, if the number of precedence constraints becomes much larger ($P = 0.2$), the problem becomes easier to solve because the processing order is restrictive. It seems that the hardest instances are those with $P = 0.05$.

7 Conclusion

This study constructed an exact algorithm for the precedence-constrained single-machine scheduling problem by extending our previous algorithms for the unconstrained problem. Numerical experiments showed that it can solve all the instances with 50 jobs of $1|prec|\sum w_i T_i$ and $1|prec|\sum(\alpha_i E_i + \beta_i T_i)$, and most instances with 100 jobs of $1|prec|\sum w_i T_i$, although CPU time is not so impressive for larger instances. To improve the algorithm, it seems necessary to reduce the size of the graph more by improving the lower bound via imposing some new constraints on the relaxations. It will also be necessary to extend the algorithm for the problem with machine idle time permitted. These are left for future research.

Acknowledgements This work has been partially supported by Grant-in-Aid for Young Scientists (B) 19760273, from the Ministry of Education, Culture, Sports, Science and Technology Japan.

References

1. J.B. Sidney, Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs, *Operations Research*, Vol. 23, pp. 283–298 (1975).
2. E.L. Lawler, Sequencing jobs to minimize total weighted completion time subject to precedence constraints, *Annals of Discrete Mathematics*, Vol. 2, pp. 75–90 (1978).
3. C.N. Potts, A Lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time, *Management Science*, Vol. 31, pp. 1300–1311 (1985).
4. J.A. Hoogeveen and S.L. van de Velde, Stronger Lagrangian bounds by use of slack variables: applications to machine scheduling problems, *Mathematical Programming*, Vol. 70, pp. 173–190 (1995).

Table 1 Results for benchmark instances of $1|prec|\sum w_i T_i$

n		P						
		0.0	0.005	0.01	0.02	0.05	0.1	0.2
40	ave	0.09	0.74	1.18	4.82	17.97	28.44	4.14
	max	0.27	3.84	18.35	132.12	84.81	96.30	29.79
	opt	125	125	125	125	125	125	125
50	ave	0.19	1.78	5.56	18.23	108.32	101.65	9.92
	max	0.43	26.98	52.77	179.27	1664.26	447.74	64.85
	opt	125	125	125	125	125	125	125
100	ave	2.75	43.58	171.82	807.97	1650.13	397.72	14.49
	max	7.48	516.30	7085.22	19638.77	16800.01	4329.91	78.08
	opt	125	125	125	121	114	125	125

Table 2 Results for benchmark instances of $1|prec|\sum(\alpha_i E_i + \beta_i T_i)$

n		P						
		0.0	0.005	0.01	0.02	0.05	0.1	0.2
40	ave	0.12	0.92	2.45	7.97	41.65	51.73	9.00
	max	0.22	5.21	22.32	98.09	195.39	312.76	33.32
	opt	125	125	125	125	125	125	125
50	ave	0.24	3.80	12.46	42.66	312.16	221.67	19.69
	max	0.49	83.24	105.03	262.31	6624.25	1344.17	72.71
	opt	125	125	125	125	125	125	125

5. S.L. van de Velde, Dual decomposition of a single-machine scheduling problem, *Mathematical Programming*, Vol. 69, pp. 413–428 (1995).
6. C. Chekuri and R. Motwani, Precedence constrained scheduling to minimize sum of weighted completion times on a single machine, *Discrete Applied Mathematics*, Vol. 98, pp. 29–38 (1999).
7. F. Margot, M. Queyranne and Y. Wang, Decompositions, network flows, and a precedence constrained single-machine scheduling problem, *Operations Research*, Vol. 51, pp. 981–992 (2003).
8. S. Tanaka, S. Fujikuma and M. Araki, An Exact Algorithm for Single-Machine Scheduling without Machine Idle Time, *Journal of Scheduling*, *in press*, DOI: 10.1007/s10951-008-0093-5.
9. S. Tanaka and S. Fujikuma, An efficient exact algorithm for general single-machine scheduling with machine idle time, 4th IEEE Conference on Automation Science and Engineering, pp. 371–376 (2008).
10. S. Tanaka and S. Fujikuma, A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time, *submitted*.
11. T. Ibaraki, Enumerative approaches to combinatorial optimization, *Annals of Operations Research*, Vols. 10 and 11 (1987).
12. T. Ibaraki and Y. Nakamura, A dynamic programming method for single machine scheduling, *European Journal of Operational Research*, Vol. 76, pp. 72–82 (1994).
13. T.S. Abdul-Razaq and C.N. Potts, Dynamic programming state-space relaxation for single-machine scheduling, *Journal of the Operational Research Society*, Vol. 39, pp. 141–152 (1988).
14. C.N. Potts and L.N. Van Wassenhove, A branch and bound algorithm for the total weighted tardiness problem, *Operations Research*, Vol. 33, pp. 363–377 (1985).
15. F. Sourd, New exact algorithms for one-machine earliness-tardiness scheduling, *INFORMS Journal on Computing*, Vol. 21, pp. 167–175 (2009).
16. J. Carlier and E. Pinson, An algorithm for solving the job-shop problem, *Management Science*, Vol. 35, pp. 164–176 (1989).
17. J. Carlier and E. Pinson, A practical use of Jackson's preemptive schedule for solving the job shop problem, *Annals of Operations Research*, Vol. 26, pp. 269–287 (1990).
18. P. Brucker, B. Jurisch, B. Sievers, A branch and bound algorithm for the job-shop scheduling problem, *Discrete Applied Mathematics*, Vol. 49, pp. 107–127 (1994).
19. P. Baptiste, C. Le Pape and W. Nuijten, *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, pp. 21–29, Kluwer Academic Publishers (2001).
20. H.D. Sherali and O. Ulular, A primal-dual conjugate subgradient algorithm for specially structured linear and convex programming problems, *Applied Mathematics and Optimization*, Vol. 20, pp. 193–221 (1989).
21. R.K. Congram, C.N. Potts and S.L. van de Velde, An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem, *INFORMS Journal on Computing*, Vol. 14, pp. 52–67 (2002).
22. A. Grosso, F. Della Croce and R. Tadei, An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem, *Operations Research Letters*, Vol. 32, pp. 68–72 (2004).

Minimizing the number of tardy jobs in single machine scheduling using MIP

S. Ourari · C. Briand · B. Bouzouia.

Abstract This paper considers the problem of scheduling n jobs on a single machine. A fixed processing time and an execution interval are associated with each job. Preemption is not allowed. The objective is to find a feasible job sequence that minimizes the number of tardy jobs. On the basis of an original mathematical integer programming formulation, this paper shows how both good-quality lower and upper bounds can be computed. Numerical experiments on Baptiste et al.'s instances are provided, which demonstrate the efficiency of the approach.

1 Introduction

A single machine scheduling problem (SMSP) consists of a set V of n jobs to be sequenced on a single disjunctive resource. The interval $[r_j, d_j]$ defines the execution window of each job j , where r_j is the release date of j and d_j , its due-date. The processing time p_j of j is known and preemption is not allowed. A job sequence σ is said feasible if, for any job $j \in V$, $s_j \geq r_j$ and $s_j + p_j \leq d_j$, s_j being the earliest starting time of Job j in σ .

In this paper, we take an interest in finding a job sequence that minimizes the number of late jobs, problem referred to as $1|r_j|\sum U_j$ in the litterature, where U_j is set to 1 if job j is late, *i.e.*, $U_j \leftarrow (s_j + p_j > d_j)$. In the sequel, we review some important papers that deal with this problem. Nevertheless, the litterature is also rich of papers that consider other important variants by considering various additional assumptions,

Samia Ourari

CDTA, lotissement du 20 août 1956, Baba Hassen, Alger, Algeria

CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France

Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France

E-mail: sourari@laas.fr

Cyril Briand

CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France

Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ; F-31077 Toulouse, France

E-mail: briand@laas.fr

Brahim Bouzouia

CDTA, lotissement du 20 août 1956, Baba Hassen, Alger, Algeria

E-mail: bouzouia@cdta.dz

such as: jobs are weighted (problem $1|r_j|\sum w_j U_j$), setup times are considered, there exists several identical machines, *etc.* For a more extended review, the reader is referred to [5].

Determining whether it exists a feasible sequence, *i.e.*, all jobs meet their due dates, is NP-complete [14]. The problem of minimizing the number of late jobs is also NP-hard [10]. Efficient branch-and-bound procedures are reported in [2, 8, 5] that solve problem instances with up to 200 jobs.

When additional assumptions are made, $1|r_j|\sum U_j$ problems can become solvable in polynomial time. For instance, when release dates are equal, $1||\sum U_j$ problems can be solved in $O(n \log(n))$ using Moore's well known algorithm [15]. Considering the case where release and due dates of jobs are similarly ordered, *i.e.*, $r_i < r_j \Rightarrow d_i \leq d_j$, Kise, Ibaraki and Mine proposed a dynamic programming algorithm that runs in $O(n^2)$ [11]. Under this same assumption, an $O(n \log(n))$ algorithm was later proposed by Lawler in 1982 [12]. Lawler [13] also described an $O(n \log(n))$ algorithm that works on preemptive nested problems $1|r_j, nested, pmtn|\sum U_j$, *i.e.*, job preemption is allowed and job execution windows are nested: $r_i < r_j \Rightarrow d_i \geq d_j$ or $d_i > d_j \Rightarrow r_i \leq r_j$. More recently, considering the general preemptive problem $1|r_j, pmtn|\sum U_j$, Baptiste designed an algorithm that runs in $O(n^4)$ [1]. When processing times are equal, Carlier [6] proposed in the early eighties a $O(n^3 \log(n))$ procedure. Nevertheless, this procedure has recently been proved non-optimal by Chrobak et al. [7] who exhibit a new optimal $O(n^5)$ algorithm.

This paper presents how, using an original Mathematical Integer Programming (MIP) formulation, both good-quality lower and upper bounds can be computed for the $1|r_j|\sum U_j$ problem. The proposed approach mainly differs from the branch-and-bound approaches described in [2], [8] and [5] in the fact that, since MIP is used, it is more generic: new constraints can easily be added to the model. Moreover, from the best of our knowledge, there does not exist other MIP approach for the $1|r_j|\sum U_j$ problem that can be compared in terms of efficiency with the already existing branch-and-bound methods.

The paper is structured as follows. First, a dominance theorem is recalled that stands for the problem of finding a feasible sequence to the SMSP. In Section 3, a MIP formulation for searching a feasible job sequence is described. The fourth section discusses the validity of the dominance theorem of Section 2 when the $\sum U_j$ criterion is considered. Section 5 and 6 show how the MIP of Section 3 can easily be adapted for computing an upper bound and a lower bound to the $1|r_j|\sum U_j$ problem. The last section is devoted to the synthesis of the numerical experiments and discusses the efficiency of our MIP approach.

2 A general dominance theorem for the SMSP

In this section, some analytical dominance conditions are recalled for the SMSP. They have been originally proposed in the early eighties by Erschler et al. [9] within a theorem that is stated in the sequel. This theorem uses the notions of a *top* and a *pyramid* that are defined below. It defines a set S_{dom} of dominant job sequences, with respect to the feasibility problem, for the SMSP. Let us recall that a job sequence σ_1 dominates another job sequence σ_2 if σ_2 feasible $\Rightarrow \sigma_1$ feasible. By extension, a set of job sequences S_{dom} is said dominant if, for any job sequence $\sigma_2 \notin S_{dom}$, it exists $\sigma_1 \in S_{dom}$ such that σ_2 feasible $\Rightarrow \sigma_1$ feasible.

Characterizing a set of dominant job sequences is of interest since, when searching for a feasible job sequence, only the set of dominant sequences need to be explored. Indeed, when there does not exist any feasible sequence in the dominant set, it can be asserted that the original problem does not admit any feasible solution. This allows a significant reduction of the search space.

Definition 1 A job $t \in V$ is a top if there does not exist any other job $j \in V$ such that $r_j > r_t \wedge d_j < d_t$ (i.e., the execution window of a top does not strictly include any other execution window).

The tops are indexed in ascending order with respect to their release dates or, in case of tie, in ascending order with respect to their due dates. When both their release dates and due dates are equal, they can be indexed in an arbitrary order. Thus, if t_a and t_b are two tops then $a < b$ if and only if $(r_{t_a} \leq r_{t_b}) \wedge (d_{t_a} \leq d_{t_b})$. Let m refers to as the total number of tops.

Definition 2 Given a top t_k , a pyramid P_k related to t_k is the set of jobs $j \in V$ such that $r_j < r_{t_k} \wedge d_j > d_{t_k}$ (i.e., the set of jobs so that their execution window strictly includes the execution window of the top).

Considering the previous definition, we stress that a non-top job can belong to several pyramids. Let $u(j)$ ($v(j)$ resp.) refers to as the index of the first pyramid (the last pyramid resp.) to which Job j can be assigned.

The following theorem can now be stated. The reader is referred to [9] for its proof.

Theorem 1 The set S_{dom} of job sequences in the form:

$$\alpha_1 \prec t_1 \prec \beta_1 \prec \dots \prec \alpha_k \prec t_k \prec \beta_k \prec \dots \prec \alpha_m \prec t_m \prec \beta_m$$

where:

- t_k is the top of Pyramid P_k , $\forall k = 1 \dots m$;
- α_k and β_k are two job subsequences located at the left and the right of Top t_k respectively, such that jobs belonging to subsequence α_k are sequenced with respect to the increasing order of their r_j , and jobs belonging to β_k , are sequenced with respect to the increasing order of their d_j ;
- any non-top j is located either in subsequence α_k or β_k , for a given k such that $u(j) \leq k \leq v(j)$.

is dominant for the problem of finding a feasible job sequence.

3 A MIP formulation for finding a feasible job sequence

In this section, the problem of searching a feasible job sequence is considered and a MIP is described that has been originally introduced in [4]. It aims at determining the most dominant job sequence among the set S_{dom} in the form $\alpha_1 \prec t_1 \prec \beta_1 \prec \dots \prec \alpha_m \prec t_m \prec \beta_m$. It is formulated as follows:

$$\begin{aligned}
 \max \quad & z = \min_{k=1, \dots, m} (D_k - R_k - p_{t_k}) \\
 \text{s.t.} \quad & \left\{ \begin{array}{ll} R_k \geq r_{t_k} & , \quad \forall k \in [1 \ m] \quad (3.1) \\ R_k \geq r_i + \sum_{\{j \in P_k | r_j \geq r_i\}} p_j x_{kj}^+ & , \quad \forall k \in [1 \ m], \forall i \in P_k \quad (3.2) \\ R_k \geq R_{k-1} + \sum_{\{j \in P_{k-1}\}} p_j x_{(k-1)j}^- + p_{t_{k-1}} & \\ & + \sum_{\{j \in P_k\}} p_j x_{kj}^+ , \quad \forall k \in [2 \ m] \quad (3.3) \\ D_k \leq d_{t_k} & , \quad \forall k \in [1 \ m] \quad (3.4) \\ D_k \leq d_i - \sum_{\{j \in P_k | d_j \leq d_i\}} p_j x_{kj}^- & , \quad \forall k \in [1 \ m], \forall i \in P_k \quad (3.5) \\ D_k \leq D_{k+1} - \sum_{\{j \in P_{k+1}\}} p_j x_{(k+1)j}^+ - p_{t_{k+1}} & \\ & - \sum_{\{j \in P_k\}} p_j x_{kj}^- , \quad \forall k \in [1 \ (m-1)] \quad (3.6) \\ \sum_{k=u(i)}^{v(i)} (x_{ki}^- + x_{ki}^+) = 1 & , \quad \forall i \in P_k \quad (3.7) \\ x_{ki}^- , \quad x_{ki}^+ \in \{0, 1\} & , \quad \forall k \in [1 \ m], \forall i \in P_k \\ D_k , \quad R_k \in \mathbb{Z} & , \quad \forall k \in [1 \ m] \end{array} \right.
 \end{aligned}$$

In this MIP, the binary variable x_{ki}^+ (x_{ki}^- resp.) is set to 1 if the job i , assigned to Pyramid P_k , is sequenced in α_k (in β_k resp.), provided that (see constraint (3.7)):

- $u(i) \leq k \leq v(i)$;
- i cannot be sequenced both in α_k and β_k ;
- i cannot be assigned to several pyramids.

The integer variable R_k corresponds to the earliest starting time of Job t_k . By definition:

$$R_k = \max(r_{t_k}, \text{eft}_{t_{k-1}} + \sum_{\{j \in \alpha_k\}} p_j, \max_{i \in \alpha_k} (r_i + p_i + \sum_{\{j \in \alpha_k | i \prec j\}} p_j)) \quad (3.8)$$

where $\text{eft}_{t_{k-1}}$ is the earliest completion time of the job subsequence β_{k-1} . As the variable R_{k-1} corresponds to the earliest starting time of Job t_{k-1} , it comes that $\text{eft}_{t_{k-1}} = R_{k-1} + p_{t_{k-1}} + \sum_{j \in \beta_{k-1}} p_j$. Therefore, the constraints (3.1), (3.2) and (3.3), according to Equation (3.8), allow to determine the value of R_k .

Symmetrically, the integer variable D_k corresponds to the latest finishing time of Job t_k . By definition:

$$D_k = \min(d_{t_k}, \text{lst}_{t_{k+1}} - \sum_{\{j \in \beta_k\}} p_j, \min_{i \in \beta_k} (d_i - p_i - \sum_{\{j \in \beta_k | j \prec i\}} p_j)) \quad (3.9)$$

where $\text{lst}_{t_{k+1}}$ is the latest starting time of the job subsequence α_{k+1} . As the variable D_{k+1} corresponds to the latest finishing time of Job t_{k+1} , it comes that $\text{lst}_{t_{k+1}} = D_{k+1} - p_{t_{k+1}} - \sum_{j \in \alpha_{k+1}} p_j$. Therefore, the constraints (3.4), (3.5) and (3.6), according to Equation (3.9), give to D_k its value.

Obviously, it can be observed that the values of the R_k and D_k variables of the MIP can directly be deduced from the values of the x_{ki}^+ and x_{ki}^- binary variables. In [4], it is shown that if $z = \min_{k=1, \dots, m} (D_k - R_k - p_{t_k})$ is maximized while respecting the constraints, then the obtained sequence dominates all the others. Indeed, the authors give the proof that, for any feasible combination of the x_{ki}^+ and x_{ki}^- variables respecting the MIP constraints, a job sequence is obtained having its maximum lateness $L_{\max}$

strictly equals to $-z$. Therefore, maximizing z is strictly equivalent to minimize the maximum lateness $L_{\max}$ and it can be asserted that any sequence $\alpha_1 \prec t_1 \prec \beta_1 \prec \dots \prec \alpha_m \prec t_m \prec \beta_m$ is feasible if and only if $z = \min_{k=1, \dots, m} (D_k - R_k - p_{t_k}) \geq 0$. In the case where $z^* < 0$, there obviously does not exist any feasible sequence of n jobs for the considered problem.

4 Dominance condition for the $1|r_j|\sum U_j$ problem

In this section, the $\sum U_j$ criterion is considered. Searching optimal solution for $1|r_j|\sum U_j$ problem amounts to determine a feasible sequence for the largest selection of jobs $E \subseteq V$. Let E^* be this selection. The jobs of E^* are on time while others are late. The late jobs can be scheduled after the jobs of E^* in any order. So they do not need to be considered when searching a feasible job sequence for on-time jobs. Consequently, Theorem 1 can be applied only to the jobs belonging to E^* . There are $\sum_{k=1 \dots n} C_n^k$ possible different selections of jobs. Regarding the $\sum U_j$ criterion, the following corollary is proved.

Corollary 1 *The union of all the dominant sequences that Theorem 1 characterizes for each possible selection of jobs is dominant for the $\sum U_j$ criterion.*

Proof The proof is obvious since the union of all the sequences that Theorem 1 characterizes for any possible selection necessarily includes the dominant sequences associated with E^* , hence an optimal solution. $\square$

As already pointed out, the number of possible job selections is quite large. Nevertheless, as explained in [16], it is not necessary to enumerate all the possible job selections to get the dominant sequences. Indeed, they can be characterized using one or more *master-pyramid sequences*. The notion of a master-pyramid sequence is somewhat close to the notion of a master sequence that Dauzères-Pérès and Sevaux proposed in [8]. It allows to easily verify if a job sequence belongs to the set of dominant sequences that Theorem 1 characterizes. For building up a master-pyramid-sequence associated with a job selection $E \subseteq V$, the m_E tops and pyramids have first to be determined. Then, knowing that the set of dominant sequences is in the form $\alpha_1(E) \prec t_1(E) \prec \beta_1(E) \prec \dots \prec \alpha_k(E) \prec t_k(E) \prec \beta_k(E) \prec \dots \prec \alpha_{m_E}(E) \prec t_{m_E}(E) \prec \beta_{m_E}(E)$, it is assumed that any non-top job j is sequenced both in $\alpha_k(E)$ and $\beta_k(E)$ (these subsequences being ordered as described in Theorem 1), $\forall k$ such that $u(j) \leq k \leq v(j)$. For illustration, let us consider a problem instance with 7 jobs such that the relative order among the release and due dates of the jobs is $r_6 < r_1 < r_3 < r_2 < r_4 < d_2 < d_3 < d_4 < (r_5 = r_7) < d_6 < d_5 < d_1 < d_7$. For this example, the-master-pyramid-sequence associated with the selection $E = V$ is (tops are in bold):

$$\sigma_{\Delta}(V) = (6, 1, 3, \mathbf{2}, 3, 6, 1, 6, 1, \mathbf{4}, 6, 1, 1, \mathbf{5}, 1, \mathbf{7})$$

Any job sequence of n jobs *compatible* with $\sigma_{\Delta}(V)$ belongs to the set of dominant sequences. A sequence s is said compatible with the master-pyramid sequence $\sigma_{\Delta}(V)$ if the order of the jobs in s does not contradict the possible orders defined by $\sigma_{\Delta}(V)$, this will be denoted as $s \in \sigma_{\Delta}(V)$. Under the hypothesis that all tops are on-time, it is obvious that $\sigma_{\Delta}(V)$ also characterizes the set of dominant sequences (according

to the $\sum U_j$ criterion) of any job selection E such that $\{t_1, \dots, t_m\} \subseteq E$. Indeed, the master-pyramid sequence $\sigma_\Delta(E)$ associated with such a selection is necessarily compatible with the master-pyramid sequence $\sigma_\Delta(V)$, *i.e.*, if s is a job sequence such that $s \in \sigma_\Delta(E)$ then $s \in \sigma_\Delta(V)$.

Nevertheless, $\sigma_\Delta(V)$ does not necessarily characterize all the job sequences being dominant for the $\sum U_j$ criterion. This assertion can easily be illustrated considering a problem V with 4 jobs having the interval structure represented on Figure 1. Job a is the top of the structure and the master-pyramid sequence $\sigma_\Delta(V)$ is (d, b, c, a, b, c, d) . Now, let us imagine that a is not selected (it is late and its interval can be ignored). In this case, there are two tops b and c and the master-pyramid sequence $\sigma_\Delta(V \setminus \{a\})$ is (d, b, d, c, d) . As it can be observed, $\sigma_\Delta(V \setminus \{a\})$ is not compatible with $\sigma_\Delta(V)$ since, in the former, Job d cannot be sequenced between b and c , while it is possible in the latter. This simple example shows that the complete characterization of the set of dominant sequences requires to enumerate all the non-compatible master-pyramid sequences, their number being possibly exponential in the worst case.

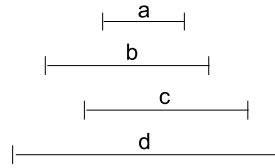


Fig. 1 An interval structure

From now the focus is on a particular SMSP where any pyramid $P_k, \forall k = 1, \dots, m$ is said *perfect*, *i.e.*, $\forall (i, j) \in P_k \times P_k, (r_i \geq r_j) \Leftrightarrow (d_i \leq d_j)$, *i.e.*, the execution intervals of the jobs belonging to P_k are included each inside the other. By extension, when all the pyramids are perfect, the corresponding SMSP will be said perfect. For this special case, the following theorem is proved:

Theorem 2 *Given a perfect SMSP V , the master-pyramid sequence $\sigma_\Delta(V)$ characterizes the complete set of sequences being dominant for the $\sum U_j$ criterion.*

Proof Obviously, removing a job j from a perfect SMSP V produces a new perfect SMSP $V \setminus \{j\}$. The proof goes by showing that the master pyramid sequence $\sigma_\Delta(V \setminus \{j\})$ is compatible with $\sigma_\Delta(V)$ (in other words, all the sequences that are compatible with $\sigma_\Delta(V \setminus \{j\})$ are also compatible with $\sigma_\Delta(V)$). Let us assume first that the removed job j is a non-top job. Since $\sigma_\Delta(V)$ is built up according to the position of the tops, removing j from V induces to remove j from all the subsequences α_k, β_k of $\sigma_\Delta(V)$ such that $u(j) \leq k \leq v(j)$ and, in this case, the relation $\sigma_\Delta(V \setminus \{j\}) \in \sigma_\Delta(V)$ necessarily holds.

Let us assume now that j is a top having the index x (*i.e.*, $\sigma_\Delta(V) = (\alpha_1, t_1, \dots, t_{x-1}, \beta_{x-1}, \alpha_x, j, \beta_x, \alpha_{x+1}, t_{x+1}, \dots, t_m, \beta_m)$). Two cases have to be considered: if j is a top such that $\forall i \in P_x \Rightarrow i \in P_{x-1}$ or $i \in P_{x+1}$, then $\sigma_\Delta(V \setminus \{j\}) = (\alpha_1, t_1, \dots, t_{x-1}, \beta_{x-1}, \alpha_{x+1}, t_{x+1}, \dots, t_m, \beta_m)$ and in this case, $\sigma_\Delta(V \setminus \{j\})$ is obviously compatible with $\sigma_\Delta(V)$. Otherwise, let k be the last job of α_x (*i.e.*, $\alpha_x = (\alpha'_x, k)$). Since the execution intervals of the jobs belonging to P_x are included each inside the other, the order of the jobs in α_x matches the reverse order

of the jobs of β_x , therefore k is also the first job of β_x (*i.e.*, $\beta_x = (k, \beta'_x)$). Then $\sigma_\Delta(V \setminus \{j\}) = (\alpha_1, t_1 \dots t_{x-1}, \beta_{x-1}, \alpha'_x, k, \beta'_x, \alpha_{x+1}, t_{x+1} \dots t_m, \beta_m)$ and in this case, $\sigma_\Delta(V \setminus \{j\})$ is obviously compatible with $\sigma_\Delta(V)$. $\square$

5 Computing an upper bound for the $1|r_j| \sum U_j$ problem

In this section, we take an interest in finding a feasible job sequence in the form $\alpha_1 \prec t_1 \prec \beta_1 \dots \prec \alpha_m \prec t_m \prec \beta_m$ that minimizes the $\sum U_j$ criterion, knowing that a non-top j is not necessarily sequenced, that is the major difference with Section 3. As discussed in Section 4, the sequences in this form are dominant for the $\sum U_j$ criterion only under the hypothesis that the top jobs are scheduled on time. As it does not necessarily exist any optimal sequence satisfying this assumption, finding an optimal sequence in this desired form only gives an upper bound to the general $1|r_j| \sum U_j$ problem. Nevertheless, as shown by the experiments (see Section 7), this upper bound is good and even often optimal.

The MIP formulation of Section 3 can easily be adapted for solving the previous problem:

$$\begin{aligned} \min \quad & z = \sum_{\{j \in V \setminus \{t_1 \dots t_m\}\}} (1 - \sum_{k=u(j)}^{v(j)} (x_{jk}^- + x_{jk}^+)) + \sum_{k=1}^m y_{t_k} \\ \text{s.t.} \quad & \left\{ \begin{array}{ll} R_k \geq r_{t_k} & , \quad \forall k \in [1 \ m] \quad (5.1) \\ R_k \geq r_i + \sum_{\{j \in P_k | r_j \geq r_i\}} p_j x_{kj}^+ & , \quad \forall k \in [1 \ m], \forall i \in P_k \quad (5.2) \\ R_k \geq R_{k-1} + \sum_{\{j \in P_{k-1}\}} p_j x_{(k-1)j}^- + p_{t_{k-1}} & \\ \quad + \sum_{\{j \in P_k\}} p_j x_{kj}^+ & , \quad \forall k \in [2 \ m] \quad (5.3) \\ D_k \leq d_{t_k} & , \quad \forall k \in [1 \ m] \quad (5.4) \\ D_k \leq d_i - \sum_{\{j \in P_k | d_j \leq d_i\}} p_j x_{kj}^- & , \quad \forall k \in [1 \ m], \forall i \in P_k \quad (5.5) \\ D_k \leq D_{k+1} - \sum_{\{j \in P_{k+1}\}} p_j x_{(k+1)j}^+ - p_{t_{k+1}} & \\ \quad - \sum_{\{j \in P_k\}} p_j x_{kj}^- & , \quad \forall k \in [1 \ (m-1)] \quad (5.6) \\ \sum_{k=u(i)}^{v(i)} (x_{ki}^- + x_{ki}^+) \leq 1 & , \quad \forall i \in P_k \quad (5.7) \\ D_k - R_k \geq p_{t_k} (1 - y_{t_k}) & , \quad \forall k \in [1 \ m] \quad (5.8) \\ y_{t_k} \quad , \quad x_{ki}^- \quad , \quad x_{ki}^+ \in \{0, 1\} & , \quad \forall k \in [1 \ m], \forall i \in P_k \\ D_k \quad , \quad R_k \in \mathbb{Z} & , \quad \forall k \in [1 \ m] \end{array} \right. \end{aligned}$$

Let us comment this MIP. First, constraints (5.1)-(5.6) are identical to constraints (3.1)-(3.6) since integer variables R_k and D_k are determined in the same way. Allowing a non-top job to be late is easy by relaxing constraint (3.7), replacing it by constraint (5.7). As the feasibility of the obtained sequence is required, the constraint $D_k - R_k \geq p_{t_k}$ is set, $\forall k = 1, \dots, m$. Nevertheless, we observe that this constraint is a bit too strong since, when two consecutive tops t_k and t_{k+1} are such that $d_{t_{k+1}} - r_{t_k} < p_{t_{k+1}} + p_{t_k}$, the MIP is unfeasible (*i.e.*, there does not exist any feasible sequence that keeps both t_k and t_{k+1} on time). For avoiding this unfeasibility, the binary variable y_{t_k} is introduced (see constraint (5.8)): y_{t_k} equals 1 if the processing time of t_k is ignored, 0 otherwise. Therefore, the $\sum U_j$ criterion can easily be expressed using the binary variables y_{t_k} ,

x_{ki}^+ and x_{ki}^- since, if $y_{t_k} = 1$, Top t_k is late and, if $\sum_{k=u(j)}^{v(j)} (x_{jk}^- + x_{jk}^+) = 0$, non-top job j is late.

6 Lower-bound for the $1|r_j|\sum U_j$ problem

In a minimization problem, a lower bound is obtained by relaxing some constraints and optimally solving the relaxed problem. According to Theorem 2, when all the pyramids are perfect, the set of sequences in the form $\alpha_1 \prec t_1 \prec \beta_1 \cdots \prec \alpha_m \prec t_m \prec \beta_m$ is dominant for the $\sum U_j$ criterion. Moreover, for any problem, we know that it is always possible to decrease the r_j values (or increase the d_j values) of some jobs in order to make the pyramids perfect, *i.e.*, such that $\forall (i, j) \in P_k \times P_k, (r_i \geq r_j) \Leftrightarrow (d_i \leq d_j)$, $\forall k = 1, \dots, m$ (see Figure 2). Doing so, a relaxed problem is obtained that can be optimally solved by the following MIP:

$$\begin{aligned} \min \quad & z = \sum_{\{j \in V \setminus \{t_1 \dots t_m\}\}} (1 - \sum_{k=u(j)}^{v(j)} (x_{jk}^- + x_{jk}^+)) + \sum_{k=1}^m y_{t_k} \\ \text{s.t.} \quad & \left\{ \begin{array}{ll} R_k \geq r_{t_k} + \mathbf{y}_{t_k}(r_{n_k} - r_{t_k}) & , \quad \forall k \in [1 \ m] \quad (6.1) \\ R_k \geq r_i + (1 - x_{ik}^+)(r_{n_k} - r_i) & \\ & + \sum_{\{j \in P_k | r_j \geq r_i\}} p_j x_{kj}^+ , \quad \forall k \in [1 \ m], \forall i \in P_k \quad (6.2) \\ R_k \geq R_{k-1} + \sum_{\{j \in P_{k-1}\}} p_j x_{(k-1)j}^- + p_{t_{k-1}} & \\ & + \sum_{\{j \in P_k\}} p_j x_{kj}^+ , \quad \forall k \in [2 \ m] \quad (6.3) \\ D_k \leq d_{t_k} + \mathbf{y}_{t_k}(d_{n_k} - d_{t_k}) & , \quad \forall k \in [1 \ m] \quad (6.4) \\ D_k \leq d_i + (1 - x_{ik}^-)(d_{n_k} - d_i) & \\ & - \sum_{\{j \in P_k | d_j \leq d_i\}} p_j x_{kj}^- , \quad \forall k \in [1 \ m], \forall i \in P_k \quad (6.5) \\ D_k \leq D_{k+1} - \sum_{\{j \in P_{k+1}\}} p_j x_{(k+1)j}^+ - p_{t_{k+1}} & \\ & - \sum_{\{j \in P_k\}} p_j x_{kj}^- , \quad \forall k \in [1 \ (m-1)] \quad (6.6) \\ \sum_{k=u(i)}^{v(i)} (x_{ki}^- + x_{ki}^+) \leq 1 & , \quad \forall i \in P_k \quad (6.7) \\ D_k - R_k \geq p_{t_k}(1 - y_{t_k}) & , \quad \forall k \in [1 \ m] \quad (6.8) \\ y_{t_k} , \quad x_{ki}^- , \quad x_{ki}^+ \in \{0, 1\} & , \quad \forall k \in [1 \ m], \forall i \in P_k \\ D_k , \quad R_k \in \mathbb{Z} & , \quad \forall k \in [1 \ m] \end{array} \right. \end{aligned}$$

with:

- $r_{n_k} = \min_{\{j \in P_k\}} r_j$;
- $d_{n_k} = \max_{\{j \in P_k\}} d_j$;

This MIP differs from the one of Section 5 only by the addition of the terms in bold that allow to deactivate the constraints of type (6.1), (6.2), (6.4) or (6.5) when some jobs are late. For instance, if t_k is late, *i.e.*, $y_{t_k} = 1$, the term $y_{t_k}(r_{n_k} - r_{t_k})$ ($y_{t_k}(d_{n_k} - d_{t_k})$ resp.) deactivates the constraint (6.1) (the constraint (6.4) resp.). Indeed, we know that the inequality $R_k \geq r_{n_k}$ (resp. $D_k \leq d_{n_k}$) is obviously always verified. Similarly, in the case where $i \notin \alpha_k$ ($i \notin \beta_k$ resp.), the term $(1 - x_{ik}^+)(r_{n_k} - r_i)$ ($(1 - x_{ik}^-)(d_{n_k} - d_i)$ resp.) deactivates the constraint (6.2) (the constraint (6.5) resp.). Note that the deactivation

of constraints allows to ensure that only the constraints that concern the on-time jobs are taken into account.

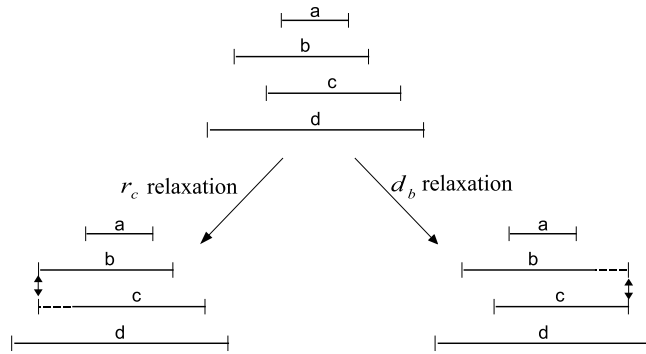


Fig. 2 Two relaxation strategies for turning a problem into a perfect one

7 Numerical experiments

For evaluating the performances of our MIP models, Baptiste et al.'s problem instances have been used (see [2]). For $n \in \{80, 100, 120, 140\}$, 120 problem instances are provided and, for each of them, thanks to the authors who provide us with their detailed results, either the optimal $\sum U_j$ value (*OPT*) or, at least, an upper-bound of this value (*BEST*), is known. For each problem instance, using a commercial MIP solver, we determined:

- two lower bounds by solving the MIP corresponding to the perfect SMSP obtained either by relaxation of the r_i (LBr_i) or by relaxation of the d_i (LBd_i), as explained in Section 6 ;
- one upper bound (*UB*) by solving the MIP described in Section 5.

In each experiment, the CPU time has been bounded to one hour. Table 1 displays, for the three kinds of MIP, the percentages of instances that were optimally solved within one hour, as well as the min / mean / max CPU time. For instance, when $n = 80$, the solver returns the optimal LBr_i value in 94.16% of the cases, with a min / mean / max CPU time of 0.01/42.35/1395.4 seconds respectively. The values of Table 1 can be compared with the percentage of time the Baptiste et al.'s method finds an optimal solution in less than one hour (see Table 3).

A few observations can be made at this point. First, even if some problem instances seem very hard to solve, optimal solutions are found in most of the cases. The computation of the upper bound is globally less time expensive than the one of the lower bound. This is not surprising since in the former case, because tops are assumed to be on time, the search space is less extended than in the latter case, where any job can be late or on time. We also observe in our experiments that the lower bound given by the relaxation of the d_i is often better than the one obtained by relaxation of the r_i . Since the two kinds of relaxation are symmetric, this is possibly due to the way problem instances have been generated.

Instances		$LB r_i$	$LB d_i$	UB
$n=80$	N	94.16%	96.66%	98.33%
	$Tcpu$	(0.01; 42.35; 1395.54) s	(0.02; 61.15; 2363.76) s	(0.02; 27.03; 1757.48) s
$n=100$	N	82.50%	81.66%	94.13%
	$Tcpu$	(0.02; 141.15; 3531.11) s	(0.03; 85.70; 1318.86) s	(0.02; 33.84; 1778.42) s
$n=120$	N	80.83%	84.16%	85%
	$Tcpu$	(0.02; 106.78; 1340.29) s	(0.04; 108.43; 2149.83) s	(0.02; 127.67; 3529.96) s
$n=140$	N	65.83%	65.00%	73.33%
	$Tcpu$	(0.05; 139.77; 1490.64) s	(0.03; 173.97; 3072.82) s	(0.02; 134.62; 2600.95) s

Table 1 Percentages of MIP solved to optimality

Instances	$LB=UB$		$UB - LB$
	All instances	$Tcpu < 3600s$	(min. mean. max)
$n=80$	70.83%	73.21%	(0; 0.38; 6)
$n=100$	70%	73.62%	(0; 0.39; 2)
$n=120$	56.66%	60.68%	(0; 0.5; 2)
$n=140$	60.83%	62.31%	(0; 0.45; 2)

Table 2 Percentages of optimal solutions

Table 2 takes an interest in the cases where the solution is optimal, *i.e.*, the upper bound equals one of the lower bounds ($UB = LB = \max(LBr_i, Lbd_i)$). Two sub-cases are distinguished according if one considers the total set of instances or only the instances for which the CPU time is lower than one hour. As one can see, optimality can be proved in many cases, even if the Baptiste et al.'s ad-hoc approach remains better from this point of view. Let us point out that our MIP approach proves the optimality of 3 instances that were not optimally solved by Baptiste et al.. The table also indicates the min / mean / max difference between the upper bound and the best lower bound: it is always small even for the largest instances.

Lastly, Table 3 gives a more tightened analysis of the quality of our upper bound UB . It is compared with either the optimal value OPT found by the Baptiste et al.'s method (when it is computed in less than one hour) or with the best solution $BEST$ that was returned otherwise. We observe that, nearly for all the instances, our upper-bound equals the optimal or the best solution found by the Baptiste et al.'s method. Moreover, we also observe that its computation is less time expensive in any cases. These observations seems to indicate that, in order to increase the percentage of solutions that our approach is able to certify optimal, the way to relax the problem for finding lower bounds should be improved. Mixed relaxation schemes where r_i and d_i values would be both relaxed, for instance intending to minimize the sum of their variations, could be explored.

Instances	Baptiste et al.		$Tcpu < 3600s$ and $UB=OPT$		$UB \leq BEST$
$n=80$	96.70%	117.3s	96.7%	27.03s	100.0%
$n=100$	90.00%	273.5s	90.00%	33.84s	100.00%
$n=120$	84.20%	538.2s	81.66%	127.64s	99.17%
$n=140$	72.50%	1037.3s	71.66%	134.62s	98.33%

Table 3 Analysis of the upper-bound quality

8 Conclusion

Designing MIP models for solving efficiently basic SMSPs is of interest since MIP approaches are often adaptable for dealing with new constraints or new objective. As a proof of this statement, this paper shows how an original MIP model, used for solving the $1|r_j|L_{\max}$ problem, can be adapted for dealing with the more complex $1|r_j|\sum U_j$ problem. Since the analytical dominance condition used for designing the MIP formulation of the former problem is valid for the $\sum U_j$ criterion only under some restrictions (tops are not late), only an upper bound can be computed. However, as shown by the experiments, this upper bound is optimal in most of the cases, or at least very close to the optimum. In the particular case where the considered SMSP is *perfect* (see Section 4), the paper gives a MIP model that allows to directly find the optimal $\sum U_j$ value. Since it is always possible to relax the release dates or the due dates of any SMSP in order to make it perfect, this MIP also allows to compute good lower bounds.

For the future works, we plan to investigate preprocessing methods by applying variable-fixing techniques from Integer Linear Programming. Such techniques, successfully used in several papers (see for instance [3]), allow to definitively fix the value of some binary variables, namely those of y_{t_k} , x_{ki}^+ and x_{ki}^- in our MIP, intending to tighten the search space and speed up the solving phase. We guess that these techniques will improve our approach from a computational viewpoint such that it becomes more competitive in comparison with the best existing branch and bound approaches.

References

1. Baptiste P., "Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine when processing times are equal", Journal of Scheduling, Vol 2, pp245-252 (1999).
2. Baptiste P., Peridy L and Pinson E., "A Branch and Bound to Minimize the Number of Late Jobs on a Single Machine with Release Time Constraints", European Journal of Operational Research, 144 (1), pp1-11 (2003).
3. Baptiste P., Della Croce F., Grosso A. and T'kindt V. (2009), "Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances", Journal Of Scheduling, ISSN 1099-1425 (Online).
4. Briand C., Ourari S. "Une formulation PLNE efficace pour $1|r_j|L_{\max}$ ", 10ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'09), Nancy (France), pp.104-105 (2009) (*in french*).
5. R. M'Hallah, R.L. Bulfin "Minimizing the weighted number of tardy jobs on a single machine with release dates", European Journal of Operational Research, 176, pp727-744 (2007).
6. Carlier J., "Problèmes d'ordonnements à durées égales", QUESTIO, 5(4), 219-228 (1981) (*in french*).
7. Chrobak M., Dürr C., Jawor W., Kowalik L., Kurowski M., "A Note on Scheduling Equal-Length Jobs to Maximize Throughput", Journal of Scheduling, Vol. 9, No 1, pp71-73 (2006).
8. Dauzère-Pérès S., Sevaux M., "An exact method to minimize the number of tardy jobs in single machine scheduling", Journal of Scheduling, Vol. 7, No 6, pp405-420 (2004).
9. Erschler, J., Fontan, G., Merce, C., Roubellat, F., "A New Dominance Concept in Scheduling n Jobs on a Single Machine with Ready Times and Due Dates", Operations Research, Vol. 31, pp114-127 (1983).
10. Michael R. Garey, David S. Johnson, "Computers and Intractability, A Guide to the Theory of NP-Completeness", W. H. Freeman and Company (1979).
11. Kise H., Toshihide I., Mine H., "A Solvable Case of the One-Machine Scheduling Problem with Ready and Due Times", Operations Research, 26(1), pp121-126 (1978).
12. Lawler E.L., "Scheduling a single machine to minimize the number of late jobs", Preprint. Computer Science Division, University of California, Berkeley (1982).

13. E.L. Lawler, "A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs", Ann. Oper. Res., 26, pp125-133 (1990).
14. Lenstra J.K., Rinnooy Han A.H.G , Brucker P., "Complexity of machine scheduling problems", Annals of Discrete Mathematics, vol. 1, pp343-362 (1977).
15. Michael J. Moore, "An n job, one machine sequencing algorithm for minimizing the number of late jobs", Management Science, 15(1), pp102-109 (1968).
16. Ourari S., Briand C. "Conditions de dominance pour le problème une machine avec minimisation des travaux en retard" 9ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'08), Clermont-Ferrand (France), pp.351-352 (2008)(*in french*).

Time Symmetry of Project Scheduling with Time Windows and Take-give Resources

Zdeněk Hanzálek · Přemysl Šůcha

Abstract The paper studies a real production scheduling problem with special resources called take-give resources. Such resources are needed from the beginning of an activity to the completion of another activity of the production process. The difference to other works is that we consider sequence dependent changeover time on take-give resources. We formulate this problem by integer linear programming and we suggest a heuristic problem solution. In the second part of the paper we discuss how to construct a schedule in backward time orientation and we formulate a time symmetry mapping. Consequently it is shown that the mapping is bijective and involutive. Finally, performance of the heuristic algorithm with the time symmetry mapping is evaluated on a set of lacquer production benchmarks.

1 Introduction

The problem that we address in this paper is motivated by a real production scheduling specifically by a lacquer production [1] which is seen as the project scheduling problem with general temporal and resource constraints. Temporal constraints between activities of a project define minimal and maximal distance between activities, e.g. a laboratory checking must be performed within a prescribed interval related to the beginning of an activity of the lacquer production procedure. The resource constraints express limits of renewable resources such as machines (e.g. lacquer mixers) or manpower (e.g. laboratory technicians). In addition, there are special resources (mixing vessels) that are needed from the beginning of an activity to the completion of another activity. This is why we extend the classical resource constrained project scheduling problem by so called *take-give resources*. Moreover we observe that each instance of the problem can be transformed to another instance of the problem which is executed

Zdeněk Hanzálek
Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague
E-mail: hanzalek@fel.cvut.cz

Přemysl Šůcha
Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague
E-mail: suchap@fel.cvut.cz

in inverse time. Based on this observation we define a time symmetry mapping and we show that the mapping is bijective and involutive.

Various types of solutions of resource constrained project scheduling problems with positive and negative time-lags have been proposed in literature. Most of exact algorithms are based on branch and bound technique [3] but this approach is suitable for problems with less than 100 activities [6]. A heuristic algorithm by Cesta et al [5] is based on constraint satisfaction problem solving. The algorithm is based on the intuition that the most critical conflicts to be resolved first are those involving activities with large resource capacity requirements. Another heuristic algorithm proposed in [14] combines the benefits of the “squeaky wheel” optimization with an effective conflict resolution mechanism, called “bulldozing”. The possibility of improving on the squeaky wheel optimization by incorporating aspects of genetic algorithms is suggested in [15]. An overview of heuristic approaches is shown in [6] where the authors compare truncated branch and bound techniques, priority rule methods and schedule improvement procedures. A beam search heuristic is applied in [6] to scheduling of rolling ingots production. This problem cover renewable resources, changeover time and batching machines.

Up to our knowledge, there is no work dealing with take-give resources in project scheduling. Scheduling with *blocking operations* [11,4] can be seen as a subproblem of scheduling with take-give resources. Operations are blocking if they must stay on a machine after finishing when the next machine is occupied by another job. During this stay the machine is blocked for other jobs, i.e. blocking operations models the absence of storage capacity between machines. On the other hand, there is a more general framework called *reservoirs* or *storage resources* [10] usually used to model limited storage capacity or inventory limits. In this framework each activity can replenish or deplete certain amount of a resource but the resource assignment is not considered. Therefore this framework can not deal for example with changeover times on this resource type required in the lacquer production problem to model mixing vessels cleaning. A similar concept, introduced in [2], are *spatial resources*. But changeover times have not been studied in this concept yet.

Our work deals with project scheduling problems with take-give resources. We formulate this problem using Integer Linear Programming (ILP) and we suggest a priority-rule based heuristic with unscheduling step. Moreover, we study time symmetry of the problem. The motivation is to improve scheduling performance for certain types of scheduling problems occurring in production scheduling. The main contributions of this paper are: a) formulation of a project scheduling problem with take-give resources, b) a heuristic solution of the problem and c) definition of time symmetry mapping, proof of its properties and optimality.

This paper is organized as follows: Section 2 describes the notation and the scheduling problem. The ILP problem formulation is shown in Section 3. Section 4 presents a heuristic algorithm for the problem with take-give resources. Time symmetry of the problem is formalized in Section 5. Section 6 presents the heuristic algorithm performance evaluation and it shows a comparison with other approaches. Section 7 concludes the paper.

2 Problem Statement

Project scheduling problem with limited renewable resources and general temporal constraints is a well established model in the research community (see [7], [9], [13]). We assume that the project deals with a set of $n + 2$ non-preemptive activities $\mathcal{V} = \{0, 1, 2, \dots, n + 1\}$. Let $p_i \in \mathbb{R}_0^+$ be a processing time of activity i and $s_i \in \mathbb{R}_0^+$ be the start time of activity i . Activities 0 and $n + 1$ with $p_0 = p_{n+1} = 0$ denote “dummy” activities which represent the project beginning and the project termination, respectively. The activities correspond to nodes of oriented graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{E}$ is a set of edges representing general temporal constraints between nodes. Each edge $e_{ij} \in \mathcal{E}$ from node i to node j is labeled by weight $\delta_{ij} \in \mathbb{R}$. The start times of activity i and activity j are subject to *temporal constraint* given by inequality

$$s_j - s_i \geq \delta_{ij} \quad \forall (i, j) \in \mathcal{V}^2; e_{ij} \in \mathcal{E}. \quad (1)$$

Let d_{ij} is the length of the longest path from node i to node j , and $d_{ij} = -\infty$ when there is no such a path or when $i = j$. Inequality $s_j - s_i \geq d_{ij}$ holds for each tuple $(i, j) \in \mathcal{V}^2$ when there is no positive cycle. When there is a positive cycle, then $d_{ii} > 0$ for all nodes i taking part in such a cycle and consequently there is no feasible schedule.

The activities are executed on a set of m (renewable) resources $\mathcal{R} = \{1, 2, \dots, m\}$. Resource $k \in \mathcal{R}$ has capacity of $R_k \in \mathbb{Z}^+$ units. Activity i requires $r_{ik} \in \mathbb{Z}_0^+$ units of resource k during its execution (i.e. from start time s_i to completion time $C_i = s_i + p_i$), such that $0 \leq r_{ik} \leq R_k$. Multiple resources may be required by one activity (sometimes called multiprocessor activity). Every activity i with $p_i = 0$ is executed on resource k with capacity $R_k = \infty$, such activity is assumed to be an event. Furthermore, we define assignment $z_{ivk} \in \{0, 1\}$, which is equal to 1 if activity i is assigned to unit v of resource k , and 0 otherwise. Consequently equation $\sum_{v=1}^{R_k} z_{ivk} = r_{ik}$ holds for each activity $i \in \mathcal{V}$ and each resource $k \in \mathcal{R}$.

In addition, we assume changeover time $o_{ij} \in \mathbb{R}_0^+$ (also called sequence dependent setup-up time) which satisfies triangular inequality. Occurrence of changeover in a given schedule is indicated by binary variable $change_{i,j,v,k}(s, z)$ which is equal to 1 if and only if $z_{ivk} = z_{jvk} = 1$ and activity i immediately precedes activity j on unit v of resource k . Consequently $s_j \geq s_i + p_i + o_{ij}^1$ holds when there exists unit v of resource k such that $change_{i,j,v,k}(s, z) = 1$.

For a given schedule (s, z) , let the set of activities in progress at time t be denoted as $\mathcal{A}(s, z, t) := \{i \in \mathcal{V}; s_i \leq t < s_i + p_i\} \cup \{i \in \mathcal{V}; \exists j \in \mathcal{V}, \exists k \in \mathcal{R}, \exists v \in \{1, \dots, R_k\}; change_{i,j,v,k}(s, z) = 1 \text{ and } s_i + p_i \leq t < s_i + p_i + o_{ij}\}$.

The *resource constraints* are defined as follows

$$\sum_{i \in \mathcal{A}(s, z, t)} r_{ik} \leq R_k \quad \forall t \in \langle 0, UB \rangle \text{ and } \forall k \in \mathcal{R}; R_k < \infty, \quad (2)$$

where UB denotes an upper bound on the schedule length [3].

The NP-hard problem of minimizing a schedule length $C_{max} = \max_{i \in \mathcal{V}} \{C_i\}$, being subject to the temporal and resource constraints, can be denoted by $PS|temp, o_{ij}|C_{max}$ in the tree-field notation of project scheduling problems (in order to avoid confusion with the start time, we use o_{ij} instead of s_{ij} proposed in [9]).

¹ The inequality holds for both immediate and non-immediate precedence, due to the satisfaction of triangular inequality, i.e. $o_{ij} \leq o_{il} + o_{lj} \quad \forall (i, l, j) \in \mathcal{V}^3$.

We extend this problem by a concept of *take-give resources* as follows. Let us assume a set of b take-give resources $\mathcal{Q} = \{1, 2, \dots, b\}$. Take-give resource $k \in \mathcal{Q}$ has capacity of $Q_k \in \mathbb{Z}^+$ units such that $Q_k < \infty$. Analogously to activity executed on resources, we introduce *occupation* executed on take-give resources. Occupation i requires $a_{ilk} \in \{0, 1\}$ units of take-give resource $k \in \mathcal{Q}$ during its execution. Occupation i starts its execution at s_i , start time of activity which takes a take-give resource, and finishes its execution at $C_l = s_l + p_l$, completion time of activity which gives back (i.e. releases) the take-give resource. Multiple take-give resources may be taken or given back by one activity, but there is at most one take-give resource taken by activity i and given back by activity l (i.e. $\sum_{k \in \mathcal{Q}} a_{ilk} \leq 1 \forall (i, l) \in \mathcal{V}^2$) and there is a path from i to l with $d_{il} > 0$. We define take-give assignment $\tilde{z}_{ivk} \in \{0, 1\}$, which is equal to 1 if occupation i is assigned to unit v of take-give resource k , and 0 otherwise. Consequently equation $\sum_{v=1}^{Q_k} \tilde{z}_{ivk} = a_{ilk}$ holds for each $(i, l) \in \mathcal{V}^2$ and each resource $k \in \mathcal{Q}$.

In the same way, as for resources, we assume changeover time $\tilde{o}_{ij} \in \mathbb{R}_0^+$ on take-give resources which also satisfies triangular inequality. Occurrence of changeover in a given schedule is indicated by binary variable $\tilde{change}_{i,j,v,k}(s, \tilde{z})$ which is equal to 1 if and only if $\tilde{z}_{ivk} = \tilde{z}_{jvk} = 1$ and occupation i immediately precedes occupation j on unit v of take-give resource k . Consequently $s_j \geq s_i + p_i + \tilde{o}_{ij}$ holds when there exists unit v of resource k such that $\tilde{change}_{i,j,v,k}(s, \tilde{z}) = 1$.

For a given schedule (s, z) , let the set of occupations in progress at time t is denoted as $\mathcal{O}(s, t, \tilde{z}) := \{(i, l) \in \mathcal{V}^2; \exists k \in \mathcal{Q}, a_{ilk} = 1, s_i \leq t < s_l + p_l\} \cup \{i \in \mathcal{V}; \exists j \in \mathcal{V}, \exists k \in \mathcal{Q}, \exists v \in \{1, \dots, Q_k\}, \tilde{change}_{i,j,v,k}(s, \tilde{z}) = 1 \text{ and } s_i + p_i \leq t < s_l + p_l + \tilde{o}_{ij}\}$. The *take-give resource constraints* are as follows:

$$\sum_{(i,l) \in \mathcal{O}(s,t)} a_{ilk} \leq Q_k \quad \forall t \in (0, UB) \text{ and } \forall k \in \mathcal{Q}. \quad (3)$$

A schedule $S = (s, z, \tilde{z})$ is feasible if it satisfies the temporal, resource and take-give resource constraints. A set of feasible schedules for given input data is denoted by $\mathcal{S}$. The problem of finding a feasible schedule with minimal C_{max} can be denoted by $PS|temp, o_{ij}, tg|C_{max}$.

An instance of problem $PS|temp, o_{ij}, tg|C_{max}$ is shown in Figure 1. The example considers 3 resources and 2 take-give resources with different capacities. There are 5 occupations, e.g. activity 12 takes resource 2 at its start time and at the completion time of activity 14 the resource is given-back. On the other hand, if we need to bind a resource taking with the completion time of an activity we could model it as it is shown in the figure on activities 2 and 3. The start time of dummy activity 3 is synchronized with the completion time of activity 2 by edges $e_{2,3}$ and $e_{3,2}$. Therefore occupation 3 can be seen as an occupation taken at the completion time of activity 2. In the same way, resource giving-back can be bind with start time of an activity as is shown in the figure on activities 10, 11. A schedule of this instance is shown in Figure 2.

3 Integer Linear Programming Formulation

In this part, we define problem $PS|temp, o_{ij}, tg|C_{max}$ by Integer Linear Programming. Let x_{ij} be a binary decision variable such that $x_{ij} = 1$ if and only if i is followed by j and $\hat{x}_{ij} = 0$ if and only if j is followed by i . Furthermore, let $\hat{y}_{ij}$ be another binary decision variable such that if $\hat{y}_{ij} = 1$ the resource constraints are eliminated in effect, i.e. activities i and j can not share an unit.

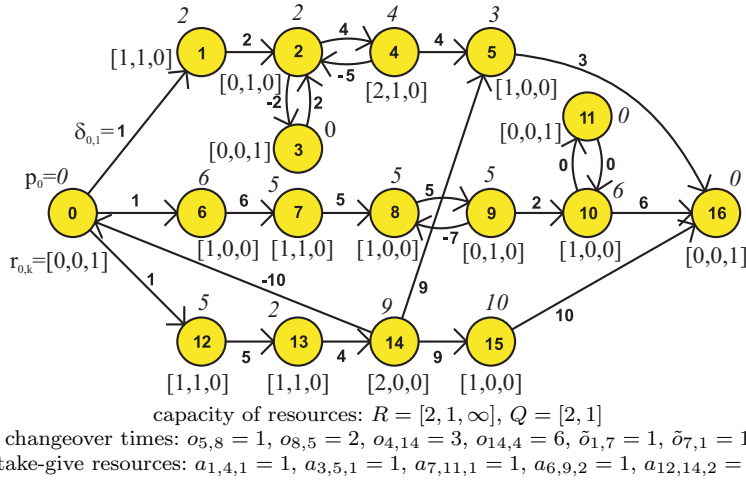


Fig. 1 An instance of problem $PS|temp, o_{ij}, tg|C_{max}$

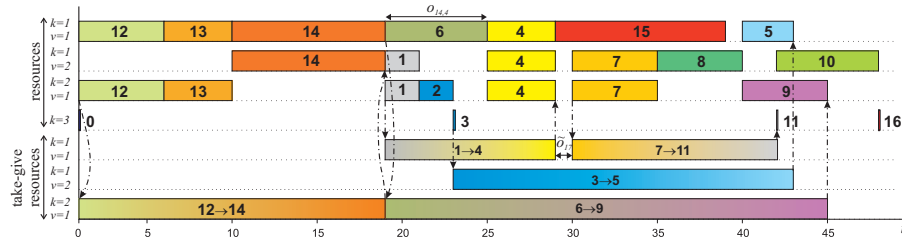


Fig. 2 A schedule of the instance from Figure 1

We define $\mathcal{M}$ as a set of possible resource conflicts. The possible resource conflict is an unordered couple $\{i, j\}$, such that $\{i, j\} \in \mathcal{M}$ iff activity i and activity j share resource k of finite capacity (i.e. $\exists k \in \mathcal{R}; r_{ik} \cdot r_{jk} \geq 1$ and $R_k < \infty$) and the execution of one activity including changeover time can overlap with the execution of the other activity (i.e. $d_{ij} < (p_i + o_{ij})$ and $d_{ji} < (p_j + o_{ji})$).

We define $\mathcal{B}$ as a set of possible take-give resource conflicts. The possible take-give resource conflict is an unordered couple $\{i, j\}$, such that $\{i, j\} \in \mathcal{B}$ iff occupation i and occupation j share take-give resource k (i.e. $\exists l \in \mathcal{V}, \exists h \in \mathcal{V}, \exists k \in \mathcal{Q}; a_{ilk} \cdot a_{jhk} = 1$) and the execution of one occupation can overlap with the execution of the other occupation (i.e. $d_{ij} < 0$ and $d_{ji} < 0$).

Constraint (5) is a direct application of temporal constraint (1). Constraints (6), (7), (8), (9) and (10) correspond to resource constraints. The binary decision variables x_{ij} and y_{ij} define the mutual relation of activities i and j . Their relation is expressed with constraints (6) and (7). There are three feasible combinations:

1. When $x_{ij} = 0$ and $y_{ij} = 0$, constraints (6) and (7) reduce to $s_j + p_j + o_{ji} \leq s_i$, i.e. j is followed by i .
2. When $x_{ij} = 1$ and $y_{ij} = 0$, constraints (6) and (7) reduce to $s_i + p_i + o_{ij} \leq s_j$, i.e. i is followed by j .
3. When $x_{ij} = 1$ and $y_{ij} = 1$, the constraints (6) and (7) are eliminated in effect and the activities i and j must be scheduled on different units.

$$\begin{aligned}
& \min s_{n+1} \tag{4} \\
\text{subject to:} & \\
& s_j - s_i \geq d_{ij}, \quad \forall (i, j) \in \mathcal{V}^2; i \neq j, d_{ij} > -\infty \tag{5} \\
& s_i - s_j + UB \cdot x_{ij} + UB \cdot y_{ij} \geq p_j + o_{ji}, \quad \forall (i, j) \in \mathcal{V}^2; i \neq j, \{i, j\} \in \mathcal{M} \tag{6} \\
& s_i - s_j + UB \cdot x_{ij} - UB \cdot y_{ij} \leq UB - p_i - o_{ij}, \quad \forall (i, j) \in \mathcal{V}^2; i \neq j, \{i, j\} \in \mathcal{M} \tag{7} \\
& -x_{ij} + y_{ij} \leq 0, \quad \forall (i, j) \in \mathcal{V}^2; i \neq j, \{i, j\} \in \mathcal{M} \tag{8} \\
& z_{ivk} + z_{jvk} - 1 \leq 1 - y_{ij}, \quad \forall (i, j) \in \mathcal{V}^2, \forall k \in \mathcal{R}, \forall v \in \{1, \dots, R_k\}; \tag{9} \\
& \quad i \neq j, \{i, j\} \in \mathcal{M}, r_{ik} \cdot r_{jk} \geq 1 \\
& \sum_{v=1}^{R_k} z_{ivk} = r_{ik}, \quad \forall i \in \mathcal{V}, \forall k \in \mathcal{R}; r_{ik} \geq 1, R_k < \infty \tag{10} \\
& \tilde{p}_i = s_l + p_l - s_i, \quad \forall (i, l) \in \mathcal{V}^2; \sum_{k \in \mathcal{Q}} a_{ilk} = 1 \tag{11} \\
& s_i - s_j + UB \cdot \tilde{x}_{ij} + UB \cdot \tilde{y}_{ij} \geq \tilde{p}_j + \tilde{o}_{ji}, \quad \forall (i, j) \in \mathcal{V}^2; i \neq j, \{i, j\} \in \mathcal{B} \tag{12} \\
& s_i - s_j + UB \cdot \tilde{x}_{ij} - UB \cdot \tilde{y}_{ij} \leq UB - \tilde{p}_i - \tilde{o}_{ij}, \quad \forall (i, j) \in \mathcal{V}^2; i \neq j, \{i, j\} \in \mathcal{B} \tag{13} \\
& -\tilde{x}_{ij} + \tilde{y}_{ij} \leq 0, \quad \forall (i, j) \in \mathcal{V}^2; i \neq j, \{i, j\} \in \mathcal{B} \tag{14} \\
& \tilde{z}_{ivk} + \tilde{z}_{jvk} - 1 \leq 1 - \tilde{y}_{ij}, \quad \forall (i, j, l, h) \in \mathcal{V}^4, \forall k \in \mathcal{Q}, \forall v \in \{1, \dots, Q_k\}; \tag{15} \\
& \quad i \neq j, \{i, j\} \in \mathcal{B}, a_{ilk} \cdot a_{jlk} = 1 \\
& \sum_{v=1}^{Q_k} \tilde{z}_{ivk} = a_{ilk}, \quad \forall (i, l) \in \mathcal{V}^2, \forall k \in \mathcal{Q}; a_{ilk} = 1 \tag{16} \\
& \text{domains of input variables are: } d_{ij} \in \mathbb{R}, p_i, o_{ij}, UB \in \mathbb{R}_0^+, r_{ik} \in \{1, \dots, R_k\}, a_{ilk} \in \{0, 1\} \\
& \text{domains of output variables are: } s_i \in \langle 0, UB - p_i \rangle, z_{ivk}, \tilde{z}_{ivk} \in \{0, 1\} \\
& \text{domains of internal variables are: } \tilde{p}_i \in \langle 0, UB \rangle, x_{ij}, \tilde{x}_{ij}, y_{ij}, \tilde{y}_{ij} \in \{0, 1\}
\end{aligned}$$

4. Combination $x_{ij} = 0$ and $y_{ij} = 1$ is not feasible due to constraint (8).

The number of units is limited using variable z_{ivk} in constraints (9) and (10). The constraint (10) satisfies that each activity i is assigned to the appropriate number of units r_{ik} for each resource $k \in \mathcal{R}$. From constraint (9) follows that when two activities i and j can overlap, i.e. $y_{ij} = 1$ then the activities can not be processed on the same unit v on resource k since $z_{ivk} + z_{jvk} - 1 \leq 0$.

The inequalities (11), (12), (13), (14), (15) and (16) stand for take-give resource constraints. The variables $\tilde{x}_{ij}$, $\tilde{y}_{ij}$ and $\tilde{z}_{ivk}$ have the same meaning as x_{ij} , y_{ij} and z_{ivk} for resource constraints. The only difference is that $\tilde{p}_i$, processing time of occupation, is a variable while p_i , processing time of activity, is a constant. Processing time $\tilde{p}_i$, expressed in equation (11) is given by s_i , start time of activity i which takes take-give resource $k \in \mathcal{Q}$ and completion time of activity l which gives back the take-give resource. Finally, the objective function of the ILP model (4) minimizes the start time of dummy activity $n + 1$, i.e. the last activity of the schedule.

4 Iterative Resource Scheduling Algorithm

The presented heuristic algorithm is a priority-rule based method with unscheduling step where activities are scheduled successively according to the given priority rule. We got the idea from iterative modulo scheduling algorithm [12] designed for iterative loops scheduling, i.e. cyclic scheduling.

For given upper bound of schedule length $C \in \langle LB, UB \rangle$, where LB is the schedule makespan lower bound and UB is the schedule upper bound, the algorithm constructs a schedule according to the priority of activities (vector *priority*). Makespan

bounds LB and UB are initialized in the same way as is described in [3]. Function $\text{findSchedule}(C, \text{priority}, \text{budget})$ tries to find a feasible schedule with $C_{max} \leq C$ while the number of scheduling steps is limited by $\text{budget} = n \cdot \text{budgetRatio}$. The algorithm parameter BudgetRatio is the ratio of the maximum number of activity scheduling steps to the number of activities n .

If function findSchedule finds a feasible schedule S , activities are shifted to the left side in function $\text{shiftLeft}(S)$ so that constraints and order of activities in S is kept. In other words, this operation finds optimal schedule S_{left} for order of activities given by S using the longest paths evaluation. Furthermore, the improved schedule S_{left} is used to update upper bound $UB = C_{max}(S_{left}) - 1$. On the other hand, if function $\text{findSchedule}(C, \text{priority}, \text{budget})$ can not find a feasible schedule within the given limit of scheduling steps (i.e. budget) the schedule lower bound is increased $LB = C + 1$. The best solution S^{best} with respect to C_{max} is searched using interval bisection method over $C \in \langle LB, UB \rangle$.

input : an instance $PS|temp, o_{ij}, tg|C_{max}, \text{budgetRatio}$
output: schedule S^{best} .
 Calculate $d_{i,j} \forall (i, j) \in \mathcal{V}^2$;
 Calculate LB and UB ;
 $C = LB$; $S^{best} = \{\}$;
 $\text{budget} = \text{budgetRatio} \cdot n$;
 $\text{priority}(i) = d_{i,n+1}$;
while $LB \leq UB$ **do**
 $S = \text{findSchedule}(C, \text{priority}, \text{budget})$;
 if S is feasible **then**
 $S_{left} = \text{shiftLeft}(S)$;
 $UB = C_{max}(S_{left}) - 1$; $S^{best} = S_{left}$;
 else
 $LB = C + 1$;
 end
 $C = \lceil (LB + UB)/2 \rceil$;
end

Algorithm 1: Iterative Resource Scheduling algorithm

Function $\text{findSchedule}(C, \text{priority}, \text{budget})$ constructs a schedule with $C_{max} \leq C$ using vector of priorities priority . As a priority we use the longest path from activity i to activity $n + 1$, i.e. $\text{priority}_i = d_{i,n+1}$. Until a feasible schedule is not found or upper bound of scheduling steps budget is not reached, the function choses such activity i with highest priority which was not scheduled yet. Subsequently, the algorithm determines the earliest and latest start time of the activity ES_i and LS_i respectively. Then function $\text{findTimeSlot}(i, ES_i, LS_i)$ finds the first $s_i \in \langle ES_i, LS_i \rangle$ such that there are no conflicts on resources $\mathcal{R}, \mathcal{Q}$. If there is no such s_i then s_i is determined according to whether activity i was scheduled once. If the activity is being scheduled for the first time, then $s_i = ES_i$ otherwise $s_i = s_i^{prev} + 1$ where s_i^{prev} is the previous start time. Finally,

function `scheduleActivity` schedule the activity at s_i . Conflicting activities, with respect to temporal or resource constraints, are unscheduled.

```

findSchedule( $C, priority, budget$ )
 $s_i = -\infty \forall i \in \{0, \dots, n+1\}$ ;
 $scheduled = \{\}$ ;

while  $budget > 0$  and  $|scheduled| < n+2$  do
     $i = \arg \max_{i \in \mathcal{V}: i \notin scheduled} (priority_i)$ ;
     $ES_i = \max_{j \in \mathcal{V}: i \in scheduled} (s_j + d_{ji})$ ;
     $LS_i = C - p_i$ ;
     $s_i = \text{findTimeSlot}(i, ES_i, LS_i)$ ;
     $scheduled = \text{scheduleActivity}(i, s_i, scheduled)$ ;
     $budget = budget - 1$ ;
end

```

Algorithm 2: Function `findSchedule`

Function `findTimeSlot` easily resolves conflicts on resource $\mathcal{R}$ but a difficulty is with take-give resources $\mathcal{Q}$ where processing time on $\mathcal{Q}$ is given by two start times. In case, that either activity i (taking resource k) or activity j (giving resource k) is not scheduled yet, the start time of the other activity is estimated using longest path d_{ij} or d_{ji} respectively. The algorithm gives very good results (see Section 6) and is flexible with respect to change of resource constraints. On the other hand, modification to another objective function (e.g. weighted tardiness $\sum w_i T_i$) is relatively difficult.

In order to improve the algorithm performance we tried to use different priority rules (shortest distance from the start, the length of a negative cycle, ...) but a priority based on the longest path gives the best results in general. We also tested a graph preprocessing while using immediate selection [3] but the gain was very small with respect to preprocessing time.

5 Time Symmetry

Let us imagine a backward execution of a given schedule of $PS|temp, o_{ij}, tg|C_{max}$ problem illustrated in Figure 3 by a “backward” time axis which is drawn below a “forward” time axis. Basically there are two ways how to construct the schedule in backward oriented time while satisfying the temporal, resource and take-give resource constraints. The first way is to change the code, i.e. re-implement a scheduling algorithm. The second way is to transform the input data and to run the original scheduling algorithm. In the rest of this section we define such transformation as *time symmetry mapping* (*TSM*) and we derive its properties with respect to $PS|temp, o_{ij}, tg|C_{max}$ problem.

Definition 1 The time symmetry is a mapping

$(a, d, o, \bar{o}, p, r, UB, s, z, \bar{z}) \xrightarrow{TSM} (a', d', o', \bar{o}', p', r', UB', s', z', \bar{z}')$ such that

$$a'_{ilk} = a_{ilk}, \quad \forall (i, l) \in \mathcal{V}^2, \forall k \in \mathcal{R} \quad (17)$$

$$d'_{ji} = d_{ij} + p_j - p_i, \quad \forall (i, j) \in \mathcal{V}^2 \quad (18)$$

$$o'_{ji} = o_{ij}, \quad \forall (i, j) \in \mathcal{V}^2 \quad (19)$$

$$\tilde{o}'_{hl} = \tilde{o}_{ij}, \quad \forall (i, j, l, h) \in \mathcal{V}^4, \exists k \in \mathcal{Q}; a_{ilk} \cdot a_{jhk} = 1 \quad (20)$$

$$p'_i = p_i, \quad \forall i \in \mathcal{V} \quad (21)$$

$$r'_{ik} = r_{ik}, \quad \forall i \in \mathcal{V}, \forall k \in \mathcal{R} \quad (22)$$

$$UB' = UB, \quad (23)$$

$$s'_i = UB - s_i - p_i, \quad \forall i \in \mathcal{V} \quad (24)$$

$$z'_{ivk} = z_{ivk}, \quad \forall i \in \mathcal{V}, \forall k \in \mathcal{R}, \forall v \in \{1, \dots, R_k\} \quad (25)$$

$$\tilde{z}'_{ivk} = \tilde{z}_{ivk}, \quad \forall i \in \mathcal{V}, \forall k \in \mathcal{R}, \forall v \in \{1, \dots, Q_k\}; a_{ilk} = 1 \quad (26)$$

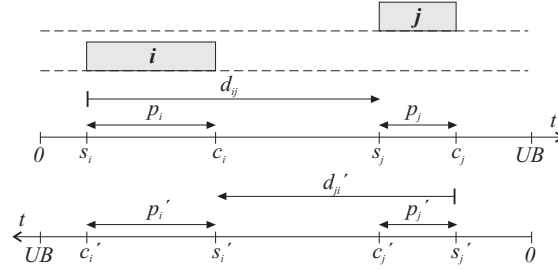


Fig. 3 Illustration of the time symmetry with forward and backward time axes

Let us assume activities i and j shown in Figure 3 where $d_{ij} > p_i$. We may observe that TSM introduces d'_{ji} with the opposite orientation (i.e. from activity j to activity i). In general variable d'_{ji} may even have a different sign than d_{ij} (for example when $d_{ij} = 1, p_j = 1, p_i = 3$) but it still preserves the same temporal constraint. Variable a in Figure 4 illustrates that the *take* operation of activity i in the forward sense is mapped on the *give* operation of activity i in the backward sense. By TSM, a matrix of change-over times is transposed and start times s'_i are given by distance from 0 on the backward time axis (i.e. UB on forward time axis).

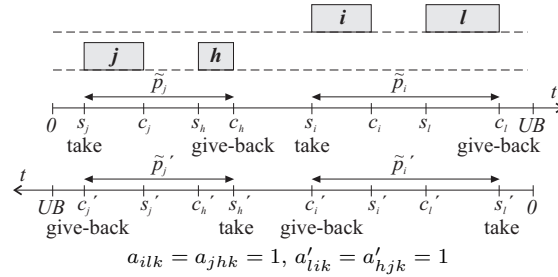


Fig. 4 Illustration of the time symmetry for take-give resources

In the following we assume that a mapping from X to Y is involutive if it holds for all $x \in X$ there exists $y \in Y$ such that $x \mapsto y \mapsto x$. For example let $X \equiv \mathbb{R}_0^+$ and $Y \equiv \mathbb{R}_0^-$, then a multiplication by -1 is involutive mapping, i.e. while applying the mapping

twice we get the identity. Let $\mathbb{D}$ be a domain of variables of $PS|temp, o_{ij}, tg|C_{max}$ problem such that $a_{ilk}, z_{ivk} \in \{0, 1\}$, $d_{ij}, s_i, p_i, o_{ij}, UB \in \mathbb{R}$, $r_{ik} \in \mathbb{Z}$ and $d_{ij} < \infty$, $0 \leq s_i \leq UB - p_i$, $0 \leq p_i \leq UB$, $0 \leq o_{ij} \leq UB$, $0 \leq \tilde{o}_{ij} \leq UB$, $UB < \infty$, $0 \leq r_{ik} \leq R_k$. The time symmetry mapping of domain $\mathbb{D}$ onto domain $\mathbb{D}$ is linear, involutive and bijective. Let $(a, d, o, \tilde{o}, p, r, UB, s, z, \tilde{z})$ are variables of $PS|temp, o_{ij}, tg|C_{max}$ problem and let $\mathbb{S}$ is the set of all feasible schedules for given n, m, b, R, Q , then time symmetry mapping of $\mathbb{S}$ onto $\mathbb{S}$ is involutive and bijective. Set $\mathbb{S}$ of feasible schedules, is equivalent to the set of feasible solutions of the system (5)-(16). By (5')-(16') we denote the system obtained by TSM from (5)-(16).

Since the time symmetry mapping of $\mathbb{S}$ onto $\mathbb{S}$ is bijective, it holds that for every feasible schedule $(a, d, o, \tilde{o}, p, r, UB, s, z, \tilde{z}) \in \mathbb{S}$, there is exactly one feasible schedule $(a', d', o', \tilde{o}', p', r', UB', s', z', \tilde{z}') \in \mathbb{S}$ such that $(a, d, o, \tilde{o}, p, r, UB, s, z, \tilde{z}) \xrightarrow{TSM} (a', d', o', \tilde{o}', p', r', UB', s', z', \tilde{z}')$. Let $(a^F, d^F, o^F, \tilde{o}^F, p^F, r^F, UB^F)$ are input data of $PS|temp, o_{ij}, tg|C_{max}$ problem and let $\mathcal{S}^F$ is the corresponding set of feasible schedules. Let $(a^B, d^B, o^B, \tilde{o}^B, p^B, r^B, UB^B)$ are input data of $PS|temp, o_{ij}, tg|C_{max}$ problem such that $(a^F, d^F, o^F, \tilde{o}^F, p^F, r^F, UB^F) \xrightarrow{TSM} (a^B, d^B, o^B, \tilde{o}^B, p^B, r^B, UB^B)$ and $\mathcal{S}^B$ is the corresponding set of feasible schedules. Time symmetry mapping from $\mathcal{S}^F$ to $\mathcal{S}^B$ is involutive and bijective.

Consequences of TSM properties: We may obtain feasible schedule $S^F = (s^F, z^F, \tilde{z}^F)$ while performing this sequence:

- we map $(a^F, d^F, o^F, \tilde{o}^F, p^F, r^F, UB^F)$ to $(a^B, d^B, o^B, \tilde{o}^B, p^B, r^B, UB^B)$
- we solve the problem by an arbitrary algorithm while obtaining $S^B = (s^B, z^B, \tilde{z}^B)$
- we map $(s^B, z^B, \tilde{z}^B)$ to $(s^F, z^F, \tilde{z}^F)$.

6 Experimental Results

In this section we show performance evaluations of Iterative Resource Scheduling algorithm. The experiments were performed on standard benchmarks UBOxxx for $PS|temp|C_{max}$ (i.e. RCPSP/max) problem (http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/) [6] and on instances of lacquers production [1, 8] where take-give resources are employed. All experiments were performed on an Intel Pentium 4 at 2.8 GHz. Algorithm Iterative Resource Scheduling was implemented in C# under Microsoft .NET framework. For all experiments with Iterative Resource Scheduling algorithm parameter was set to *budgetRatio* = 2 unless indicated otherwise.

Iterative Resource Scheduling algorithm was tested on standard benchmarks for RCPSP/max problem [6]. The results for benchmark UBO500 with $n = 500$ are summarized in Table 2 in Appendix where the first column denotes name of the benchmark instance. Column C_{max}^{best} denotes the best C_{max} obtained by algorithms: BB (Branch-and-bound algorithm), AM (Approximation method), FB (Filtered Beam Search), PR (Multi-Pass Priority-Rule Method), TS (Tabu Search) and GA (Genetic Algorithm) [6]. Name of the algorithm, which found the solution with C_{max}^{best} , is denoted in column *Algorithm*. Resulting C_{max} obtained by Iterative Resource Scheduling algorithm (C_{max}^{IRS}) is shown in the next column while the corresponding computation time is mentioned in column *CPU time*.

With respect to results obtained by [6] (considering 500 seconds time limit on 333MHz Pentium PC for each algorithm) our results are comparable or better. In addition, since Iterative Resource Scheduling algorithm solves the more general problem,

Instance	Orders			n	F <i>budgetRatio</i> = 2		F <i>budgetRatio</i> = 4		B <i>budgetRatio</i> = 2	
	met	bro	uni		C_{max} [-]	CPU time [s]	C_{max} [-]	CPU time [s]	C_{max} [-]	CPU time [s]
lacquery-1-01.ins	7	8	7	170	73681	3.58	73681	9.66	71435	13.31
lacquery-1-02.ins	8	8	7	176	77449	3.92	77449	10.41	75203	15.56
lacquery-1-03.ins	8	7	8	175	68313	4.77	68683	13.16	73432	11.66
lacquery-1-04.ins	7	8	8	178	74455	3.88	74455	9.61	72231	13.91
lacquery-1-05.ins	8	8	8	184	78223	4.72	78223	12.09	75999	16.36
lacquery-1-06.ins	9	8	8	190	81991	5.59	81991	14.31	81169	17.08
lacquery-1-07.ins	8	9	8	193	77594	2.67	77594	6.39	84621	16.08
lacquery-1-08.ins	8	8	9	192	78688	5.36	78688	14.80	77180	12.99
lacquery-1-09.ins	9	9	8	199	85718	6.94	81362	9.55	88389	18.940
lacquery-1-10.ins	9	8	9	198	82456	6.30	82456	17.20	80948	15.61
lacquery-1-11.ins	8	9	9	201	81950	6.360	81950	21.70	81866	16.88
lacquery-1-12.ins	9	9	9	207	85718	7.58	85718	25.05	85634	19.38
lacquery-1-13.ins	10	9	9	213	89486	9.03	89486	30.4	89402	21.59
lacquery-1-14.ins	9	10	9	216	90539	8.95	86409	15.00	90671	35.69
lacquery-1-15.ins	9	9	10	215	87844	8.02	87844	25.25	89316	22.24
lacquery-1-16.ins	10	10	9	222	94307	10.56	90177	19.55	94439	40.25
lacquery-1-17.ins	10	9	10	221	91612	9.19	91612	24.39	93084	26.11
lacquery-1-18.ins	9	10	10	224	90798	8.11	90798	20.31	92462	25.49
lacquery-1-19.ins	10	10	10	230	94566	9.92	94566	24.86	96230	28.45

Table 1 Lacquer production benchmarks

CPU time of the algorithm is increased by “useless” code for constraints for changeover times and take-give resources. In 10% of the instances Iterative Resource Scheduling algorithm found a better solution and similar results were obtained for set of benchmarks UBO1000 with $n = 1000$.

To evaluate influence of time symmetry on Iterative Resource Scheduling algorithm we generated 19 benchmark instances by varying the number of production orders for different lacquers [1]. Table 1 shows the results that have been obtained on these benchmark instances. The first column denotes name of the benchmark instance while corresponding number of orders is shown in column *Orders*. Subcolumn *met* states for number of metallic lacquer batches, *bro* for bronze lacquer batches and *uni* for lacquer batches of type universal. Corresponding total number of activities is indicated in column n . Remaining three columns of the table show results (i.e., C_{max} and *CPU time*) for original problem (*F* forward) with *budgetRatio* = 2 and *budgetRatio* = 4 and the problem obtained by time symmetry mapping (*B* backward) with *budgetRatio* = 2.

The Influence of algorithm parameter *budgetRatio* is demonstrated in Figure 5. This experiment was performed on a set of 90 benchmark instances UBO100 with $n = 100$. Value *meanCPU time* denotes mean algorithm computation time. The number of feasible solutions found by Iterative Resource Scheduling algorithm is denoted by *#feasible*. Up to our experiences, a reasonable compromise between computation time and quality of the resulting schedule is usually achieved with *budgetRatio* = 2.

7 Conclusions

Inspired by practical case study we extended the project scheduling problem by take-give resources which allows to consider changeover times. Furthermore, this paper proposes a heuristic problem solution suitable for considerably large problem instances. Significant part of the paper is dedicated to the time symmetry mapping, i.e. scheduling in backward time orientation. We shows that this mapping, a simple and fast polynomial transformation, is bijective and involutive.

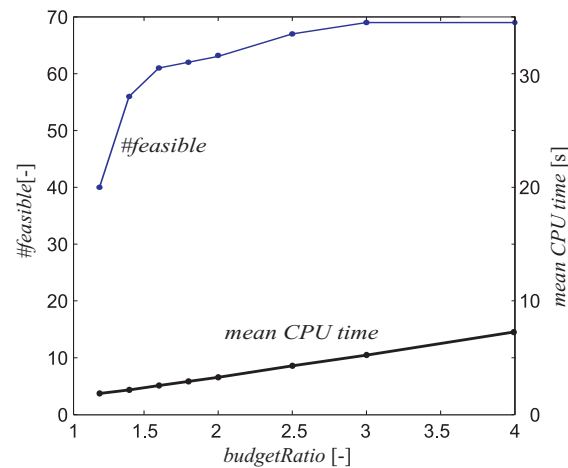


Fig. 5 Parameter *budgetRatio* influence

In order to improve the solution while using the same heuristic algorithm we used the time symmetry to find new feasible solution (in 4% of randomly generated benchmarks) and better C_{max} (about 10 % in half of the cases). A use of different priority rules and preprocessing techniques leads to considerably smaller improvements while having the same processing time.

In our current work we apply the approach described in this article to a production optimization on the assembly line. The structure of the problem is such that while using the time symmetry mapping, better C_{max} by factor of 10% is always found.

In our future work, we would like to exploit the time symmetry mapping inside heuristic algorithms while performing several iterations in forward and several in backward sense or while sharing information between two algorithms running in parallel, since the mapping is easy to extend to internal variables (like primal and dual formulation used in linear programming). In a similar manner, we would like to make use of this symmetry in Constraint Programming algorithms.

Acknowledgements The authors would like to thank Mirko Navara from Center for Machine Perception at Czech Technical University for valuable comments and inspirational suggestions.

This work was supported by the Ministry of Education of the Czech Republic under Research Programme MSM6840770038.

References

1. Gerd Behrmann, Ed Brinksma, Martijn Hendriks, and Angelika Mader. Production scheduling by reachability analysis - a case study. In *Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, page 140.1. IEEE Computer Society Press, 2005.
2. Ronald De Boer. *Resource-constrained multi-project management, a hierarchical decision support system*. PhD thesis, University of Twente, 1998.
3. Peter Brucker, Thomas Hilbig, and Johan Hurink. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 94(1-3):77–99, May 1999.
4. Peter Brucker and Thomas Kampmeyer. Cyclic job shop scheduling problems with blocking. *Annals of Operations Research*, 159(1):161–181, 2008.

5. Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, 2002.
6. Birger Franck, Klaus Neumann, and Christoph Schwindt. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR Spektrum*, 23(3):297–324, August 2001.
7. Willy Herroelen, Bert De Reyck, and Erik Demeulemeester. Resource-constrained project scheduling : A survey of recent developments. *Computers and operations research*, 25(4):279–302, 1998. Elsevier.
8. Jan Kelbel and Zdeněk Hanzálek. A case study on earliness/tardiness scheduling by constraint programming. In *Twelfth International Conference on Principles and Practice of Constraint Programming - Doctoral Program*, pages 108–113. Laboratoire DIInformatique de Nantes Atlantique, 2006.
9. Christoph Schwindt Klaus Neumann and Jrgen Zimmermann. *Project Scheduling with Time Windows and Scarce Resources*. Springer, 2003.
10. Philippe Laborie. Algorithms for propagating resource constraints in ai planning and scheduling: existing approaches and new results. *Artif. Intell.*, 143(2):151–188, 2003.
11. Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498 – 517, 2002.
12. B. Ramakrishna Rau. Iterative modulo scheduling. PROGRESS 2000 Workshop on Embedded Systems, Utrecht, The Netherlands, 2000.
13. Christoph Schwindt. *Resource Allocation in Project Management*. Springer, 2005.
14. Tristan B. Smith. An effective algorithm for project scheduling with arbitrary temporal constraints. In *Proceedings of the 19 th National Conference on Artificial Intelligence. (2004)*, pages 544–549, San Jose, California, USA, 2004. AAAI Press, Menlo Park, California.
15. Justin Terada, Hoa Vo, and David Joslin. Combining genetic algorithms with squeaky-wheel optimization. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1329–1336, New York, NY, USA, 2006. ACM Press.

A List of Variables

a_{ilk}	binary constant indicating requirement of take-give resource k for occupation i i.e. $a_{ilk} = 1$ iff take-give resource k is taken at s_i and given (released) at C_l
b	number of take-give resources
C_i	completion time of activity i
C_{max}	schedule makespan
d_{ij}	length of the longest path from node i to node j
ES_i	the earliest start time of the activity i
G	graph
i, j, l, h	activity indices
I	instance, given by input parameters of the scheduling problem
k	resource index
LB	lower bound on schedule makespan
LS_i	the latest start time of the activity i
m	number of resources
n	number of activities
o_{ij}	changeover time (sequence dependent setup-up time) from activity i to activity j
$\bar{o}_{ij}$	changeover time (sequence dependent setup-up time) from occupation i to occupation j
p_i	processing time of activity i
$\tilde{p}_i$	processing time of occupation i i.e. $\tilde{p}_i = C_l - s_i$ iff exists take-give resource k taken at s_i and given at C_l
Q_k	capacity of take-give resource k in number o units
r_{ik}	requirement of resource k for activity i
R_k	capacity of resource k in number o units
S	schedule
s_i	start time of activity i
UB	upper bound on schedule makespan
v	unit index
x_{ij}	binary variable indicating whether activity i precedes activity j i.e. $x_{ij} = 1$ iff $(s_i < s_j)$
$\tilde{x}_{ij}$	binary variable indicating whether occupation i precedes occupation j i.e. $\tilde{x}_{ij} = 1$ iff $(s_i < s_j)$
y_{ij}	binary variable indicating whether activity i and activity j must be assigned to different units of common resources i.e. if $y_{ij} = 1$ then $\forall k \in \{1, \dots, m\}, \forall v \in \{1, \dots, R_k\}; z_{ivk} \cdot z_{jvk} \neq 1$
$\tilde{y}_{ij}$	binary variable indicating whether occupation i and occupation j must be assigned to different units of common take-give resource i.e. if $\tilde{y}_{ij} = 1$ then $\forall k \in \{1, \dots, b\}, \forall v \in \{1, \dots, Q_k\}; \tilde{z}_{ivk} \cdot \tilde{z}_{jvk} \neq 1$
z_{ivk}	assignment binary variable i.e. $z_{ivk} = 1$ iff activity i is assigned to unit v of resource k
$\tilde{z}_{ivk}$	take-give assignment binary variable i.e. $\tilde{z}_{ivk} = 1$ iff occupation i is assigned to unit v of take-give resource k
δ_{ij}	length of edge e_{ij}
$\mathcal{E}$	set of edges e_{ij}
$\mathcal{M}$	set of resource conflicts between two activities i.e. unordered couple $\{i, j\} \in \mathcal{M}$ iff $\exists k \in \{1, \dots, m\}; r_{ik} \cdot r_{jk} \geq 1$ and $R_k < \infty$ and $(d_{ij} < (p_i + o_{ij})$ and $d_{ji} < (p_j + o_{ji}))$
$\mathcal{Q}$	set of take-give resources
$\mathcal{R}$	set of resources
$\mathcal{S}$	set of feasible schedules
$\mathcal{V}$	set of activities

B Results on Benchmarks

Instance	C_{max}^{best} [-]	Algorithm	C_{max}^{IRS} [-]	CPU time [s]	Instance	C_{max}^{best} [-]	Algorithm	C_{max}^{IRS} [-]	CPU time [s]
UBO500-01	inf ²	BB	inf	12,03	UBO500-46	821	AM	821	6,89
UBO500-02	2353	GA	inf	36,05	UBO500-47	1512	GA	1432	86,02
UBO500-03	2045	GA	inf	59,97	UBO500-48	1181	GA	inf	128,94
UBO500-04	2774	TS	2875	18,80	UBO500-49	1209	GA	inf	85,56
UBO500-05	inf	BB	inf	20,74	UBO500-50	1326	BB	1326	7,38
UBO500-06	2558	GA	inf	29,97	UBO500-51	1223	AM	1223	5,89
UBO500-07	inf	BB	inf	27,59	UBO500-52	1109	BB	1109	5,67
UBO500-08	2212	GA	inf	24,52	UBO500-53	1029	BB	1029	8,84
UBO500-09	inf	BB	inf	18,80	UBO500-54	825	BB	825	6,97
UBO500-10	2366	GA	inf	31,11	UBO500-55	1153	AM	1153	6,16
UBO500-11	589	AM	589	17,50	UBO500-56	976	BB	976	7,88
UBO500-12	1101	BB	1105	83,66	UBO500-57	1238	BB	1238	8,53
UBO500-13	1424	AM	1433	152,31	UBO500-58	1314	BB	1314	14,49
UBO500-14	1130	GA	inf	203,67	UBO500-59	1060	BB	1060	6,31
UBO500-15	683	GA	669	18,31	UBO500-60	1067	BB	1067	8,95
UBO500-16	982	GA	931	11,34	UBO500-61	2175	GA	2398	192,70
UBO500-17	1122	PR	1122	47,63	UBO500-62	2962	GA	3601	61,98
UBO500-18	978	TS	965	59,05	UBO500-63	inf	BB	inf	12,08
UBO500-19	1084	GA	inf	243,77	UBO500-64	2160	GA	2331	156,99
UBO500-20	1027	GA	inf	245,56	UBO500-65	inf	BB	inf	12,58
UBO500-21	717	BB	717	13,49	UBO500-66	3167	GA	inf	14,47
UBO500-22	983	BB	983	7,69	UBO500-67	2905	GA	inf	19,30
UBO500-23	848	AM	848	10,80	UBO500-68	2337	GA	2718	85,11
UBO500-24	1107	BB	1107	7,02	UBO500-69	2459	GA	2620	93,83
UBO500-25	1027	BB	1027	6,75	UBO500-70	2123	GA	inf	12,67
UBO500-26	804	BB	804	6,08	UBO500-71	1343	GA	1311	88,74
UBO500-27	749	BB	749	8,97	UBO500-72	1437	FB	1439	67,02
UBO500-28	913	BB	913	6,45	UBO500-73	1925	GA	inf	104,56
UBO500-29	893	BB	893	7,36	UBO500-74	2459	GA	inf	93,59
UBO500-30	792	BB	792	7,66	UBO500-75	976	BB	976	47,77
UBO500-31	inf	BB	inf	13,47	UBO500-76	2077	GA	2108	75,88
UBO500-32	inf	BB	inf	14,97	UBO500-77	1047	FB	1043	84,38
UBO500-33	2343	GA	inf	24,02	UBO500-78	2011	FB	2086	70,89
UBO500-34	inf	BB	inf	11,88	UBO500-79	1727	TS	1755	68,03
UBO500-35	inf	BB	inf	10,48	UBO500-80	1462	GA	1550	83,83
UBO500-36	2211	GA	2382	305,72	UBO500-81	1164	BB	1164	6,38
UBO500-37	2318	GA	inf	14,67	UBO500-82	1238	BB	1238	5,09
UBO500-38	2575	TS	inf	41,22	UBO500-83	1849	AM	1878	51,61
UBO500-39	inf	BB	inf	11,92	UBO500-84	936	BB	938	29,42
UBO500-40	2628	GA	inf	23,67	UBO500-85	1418	BB	1418	5,58
UBO500-41	1243	TS	1226	97,22	UBO500-86	1420	BB	1420	5,19
UBO500-42	1038	FB	1079	103,77	UBO500-87	1276	AM	1274	42,17
UBO500-43	1354	TS	1333	79,39	UBO500-88	1300	BB	1300	5,00
UBO500-44	801	AM	801	8,64	UBO500-89	1419	BB	1419	16,22
UBO500-45	1088	BB	1088	10,63	UBO500-90	1062	BB	1062	6,69

² Schedule was not found.

Table 2 Benchmark UBO500

Decomposed Software Pipelining for cyclic unitary RCPSP with precedence delays ^{*}

Abir Benabid · Claire Hanen

Abstract In this paper we adress a new cyclic problem: finding periodic schedules for unitary resource constrained cyclic scheduling problem. Such resource constraints are characterized by k , the number of types of functional units employed and m_x the maximal number of processors of the same type. The main problem is to cope with both precedence delays and resources which make the problem $\mathcal{NP}$ -complete in general.

A guaranteed approach, called decomposed software pipelining, has been proposed by Gasperoni and Schwiegelshohn, followed by the retiming method by Calland, Darté and Robert to solve the problem for parallel processors and ordinary precedences. We present, in this paper, an extension of this approach to resource-constrained cyclic scheduling problems with precedence delays and we provide an approximation algorithm. Let λ and λ_{opt} be respectively the period given by the algorithm and the optimal period. We establish the bound:

$$\lambda \leq \left(k + 1 - \frac{1}{m_x(\rho + 1)}\right) \lambda^{opt} + \left(1 - \frac{1}{m_x(\rho + 1)}\right) (\delta - 1).$$

1 Introduction

Cyclic scheduling problems have numerous practical application in production systems [1] as well as in embeded systems [2]. Our research in this field is particaly motivated by the advances in hardware technology, but our results still available for mass production systems.

^{*} *ORCYAE project supported by awards from DIGITEO, a research park in Ile-de-France dedicated to information science and technology.*

A. Benabid
LIP6, Pierre and Marie Curie University
E-mail: Abir.Benabid@lip6.fr

C. Hanen
Paris Ouest University and LIP6
E-mail: Claire.Hanen@lip6.fr

Embedded architectures used for devices such as mobile, automotive and consumer electronics need high performance, low silicon implementation costs, low power consumption and rapid development to ensure minimum time-to-market. Most of today's high performance applications use instruction level parallel processors such as VLIW processors [3].

VLIW architectures are mainly used for media processing in embedded devices, and instruction schedules produced by the compiler are a performance critical optimization that has a direct impact on the overall system cost and energy consumption. High-quality instruction schedules enable to reduce the operating frequency given real-time processing requirements. Most of the parallelism present in these systems is expressed in the form of loops.

In this paper, we consider a loop composed of many tasks that are to be executed a large number of times. Instruction scheduling for inner loops is also known as software pipelining [4] and can be modelised by a cyclic scheduling problem. Among the different cyclic scheduling frameworks, modulo scheduling [5] is the most successful in production compilers. The modulo scheduling focuses on finding a periodic schedule with the minimal period λ .

The classical results of modulo scheduling apply to problems that are too limited to be of practical use in instruction scheduling in modern processors as well as in mass production problems. For example, these results assume simple precedence constraints on tasks in a schedule, instead of precedences with delays like those in pipelined processors, and focus on machine models where each operation uses one of m identical processors for its execution.

In order to model the software pipelining problem, [6] proposes an extension of the classic modulo scheduling problem to resource-constrained modulo scheduling problems with precedence delays where the resources are adapted from the renewable resource of the resource-constrained scheduling problem [7].

We define, in this paper, a special case of this problem where the resource demands are unitary, and we present a guaranteed algorithm for these problems.

1.1 Problem formulation

An instance of a unitary resource-constrained cyclic scheduling problem can be defined by:

- An architecture model defined by $\mathcal{P} = \{P_{(i,j)} \mid 1 \leq i \leq k, 1 \leq j \leq m_i\}$, where k denotes the number of different types of processors and m_i denotes the number of type i processors. Let $m_x = \max_{1 \leq r \leq k} m_r$.
- A set of n tasks $V = \{T_i \mid 1 \leq i \leq n\}$ with integer processing time $\{p_i \mid 1 \leq i \leq n\}$. To each task T_i is associated a binary vector $\{b_r^i \mid 1 \leq r \leq k\}$ over the resource types, such that T_i uses b_r^i units of resource of type r during its execution. Notice that a task might use several processors but of different types.
- Each task T_i must be performed an infinite number of times. We call T_i at iteration q the q -th execution of T_i .
- Precedence graph $G(V, E)$ where:
 - E is a set of edges defining uniform dependence relations denoted by $T_i \xrightarrow{l_{ij}, h_{ij}} T_j$, where the delay l_{ij} and the height h_{ij} are nonnegative integers. l_{ij} and h_{ij}

model the fact that the task T_j at iteration q has to be issued at least l_{ij} time units after the end of task i in iteration $q - h_{ij}$.

- We denote $\rho = \frac{l_{max}}{p_{min}}$ where $l_{max} = \max_{(T_i, T_j) \in E} l_{ij}$ and $p_{min} = \min_{1 \leq i \leq n} p_i$.

Notice that this model generalize the classical parallel processors statements (in which $k = 1$ -i.e. there is a unique type of processors) as well as typed tasks systems where each binary vector $\{b_r^i\}_{1 \leq r \leq k}$ has only one positive component. Let us illustrate the notions of tasks, iterations, delays and heights with the following example. We will work on this example throughout the paper.

For $1 \leq i \leq N$

$(T_1) : t_1(i) = t_3(i)$

$(T_2) : t_2(i) = t_3(i)$

$(T_3) : t_3(i) = t_2(i - 3)$

$(T_4) : t_4(i) = t_2(i - 1)$

$(T_5) : t_5(i) = t_4(i - 2)$

$+ t_2(i - 1) + t_7(i)$

$(T_6) : t_6(i) = t_5(i - 1)$

$(T_7) : t_7(i) = t_6(i - 1)$

Tasks	T_1	T_2	T_3	T_4	T_5	T_6	T_7
p_i	2	2	3	3	1	1	1
$\{b_1^i, b_2^i\}$	1,1	1,1	1,0	1,0	1,1	0,1	1,1

The loop has $n = 7$ tasks, each one is executed N times. N is a given parameter representing the number of iterations and can be very large, and thus is assumed infinite. The associated precedence graph $G(V, E)$ is given in Figure 1. Values of l_{ij} and h_{ij} are displayed next to the corresponding arc. The resource request $\{b_1^i, b_2^i\}$ of each task T_i is highlighted next to the corresponding node.

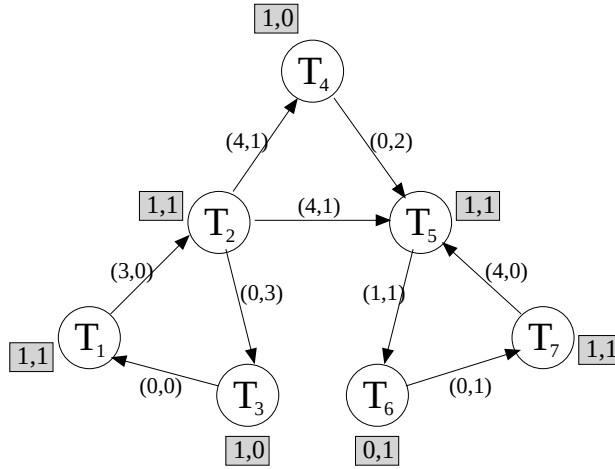


Fig. 1 An example of precedence graph $G(V, E)$.

A resource-constrained cyclic scheduling problem is to find a schedule σ that assigns a starting time $\sigma(T_i, q)$ for each task occurrence (T_i, q) such that for all $r \in \{1, \dots, k\}$

and for each time s , the number of tasks using processors of type r at time s is at most equal to m_r , and

$$\left(T_i \xrightarrow{l_{ij}, h_{ij}} T_j\right) \Rightarrow \sigma(T_i, q) + p_i + l_{ij} \leq \sigma(T_j, q + h_{ij}) \quad \forall q \in \mathbb{N}$$

The modulo scheduling focuses on finding a periodic schedule with the minimal period λ such that:

$$\forall i \in \{1, \dots, n\}, \forall q \in \mathbb{N} : \sigma(T_i, q) = \sigma(T_i, 0) + q \cdot \lambda$$

Periodic schedules are of high interest from a practical point of view, because their representation is compact so that they can be easily implemented in real systems.

1.2 Decomposed software pipelining

Generating an optimal resource constrained cyclic scheduling with minimal period is known to be $\mathcal{NP}$ -hard. To overcome this $\mathcal{NP}$ -hardness, we used the decomposed software pipelining approach introduced simultaneously by Gasperoni and Schwiegelsohn [8], and by Wang, Eisenbeis, Jourdan and Su [9]. The main idea is to decouple the problem into dependence constraints and resource constraints so as to decompose the problem into two subproblems: a cyclic scheduling problem ignoring resource constraints, and a standard acyclic graph for which efficient techniques are known.

Gasperoni and Schwiegelsohn give an efficiency bound to the period λ for the problem with m identical processors and precedences without delays. Let λ_{opt} be the optimal (smallest) period, this bound is given by the following inequality:

$$\lambda \leq \left(2 - \frac{1}{m}\right) \lambda_{opt} + \left(1 - \frac{1}{m}\right) \left(\max_{1 \leq i \leq n} p_i - 1\right)$$

Darte *et al.* [10] presents a heuristic based on circuit retiming algorithms to generalize the efficiency bound given for Gasperoni-Schwiegelsohn algorithm. The main idea is to use a retiming $\mathcal{R}$ to decide which edges to cut in $G(V, E)$ so as to make it acyclic. Then, the acyclic makespan minimization problem is solved using any approximated algorithm, that provides a pattern (schedule of a period) for a feasible periodic schedule.

In this paper, we generalize this approach for our problem. Several new elements have to be taken into account: extended resource constraints and arbitrary precedence delays.

We first use the definition of [10] for a legal retiming:

$$\mathcal{R} : V \rightarrow \mathbb{Z}, \quad \forall (T_i, T_j) \in E, \quad \mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i) \geq 0$$

A legal retiming for G of Figure 1 is given in Table 1.

Table 1 A retiming $\mathcal{R}$ of G .

Tasks	T_1	T_2	T_3	T_4	T_5	T_6	T_7
$\mathcal{R}(T_i)$	0	1	0	2	2	1	1

Then, we define the acyclic graph $G^{\mathcal{R}}$ by keeping only the arcs of G for which $\mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i) = 0$. We add two dummy tasks $Start$ and $Stop$ with null processing times and no resource use. For each task T_i , we add arcs $(Start, T_i)$ and $(T_i, Stop)$ and we define valuations of these new arcs such that:

$$\forall (T_i, T_j) \in G \setminus G^{\mathcal{R}}, l_{i, Stop} + l_{Start, j} \geq l_{ij}. \quad (1)$$

Notice that such valuations can be defined as follows for each task T_i :

$$\begin{aligned} - l_{Start, i} &= 0 \\ - l_{i, Stop} &= \max_{(T_i, T_j) \in G \setminus G^{\mathcal{R}}} l_{ij}. \end{aligned}$$

Let $\pi^{\mathcal{R}}$ be any (non cyclic) schedule of $G^{\mathcal{R}}$ that fulfills the resource constraints as well as the precedences induced by $G^{\mathcal{R}}$. We note $\pi_i^{\mathcal{R}}$ the start time of task T_i in this schedule. Then, setting $\lambda^{\mathcal{R}} = \pi_{Stop}^{\mathcal{R}}$ and for any task T_i ,

$$\sigma^{\mathcal{R}}(T_i, q) = \pi_i^{\mathcal{R}} + (q + \mathcal{R}(T_i)) \lambda^{\mathcal{R}}$$

we get the following result:

Lemma 1 $\sigma^{\mathcal{R}}$ is a feasible periodic schedule of G with period $\lambda^{\mathcal{R}}$.

Proof. First, we prove that, at any time slot t , the tasks scheduled at t in $\sigma^{\mathcal{R}}$ meet the resource constraints. We note F_t the set of tasks for which one of occurrence is scheduled at t . Let T_i and T_j be two tasks in F_t . We note q and q' their corresponding occurrences such that $\sigma^{\mathcal{R}}(T_i, q) \leq t < \sigma^{\mathcal{R}}(T_i, q) + p_i$, $\sigma^{\mathcal{R}}(T_j, q') \leq t < \sigma^{\mathcal{R}}(T_j, q') + p_j$. Hence,

$$\begin{aligned} t &= \sigma^{\mathcal{R}}(T_i, q) + s = \sigma^{\mathcal{R}}(T_j, q') + s' \\ \pi_i^{\mathcal{R}} + (q + \mathcal{R}(T_i)) \lambda^{\mathcal{R}} + s &= \pi_j^{\mathcal{R}} + (q' + \mathcal{R}(T_j)) \lambda^{\mathcal{R}} + s' \\ \pi_i^{\mathcal{R}} - \pi_j^{\mathcal{R}} + s - s' - (\mathcal{R}(T_j) - \mathcal{R}(T_i) + q' - q) \lambda^{\mathcal{R}} &= 0. \end{aligned} \quad (2)$$

Since $\pi_i^{\mathcal{R}} + s < \pi_i^{\mathcal{R}} + p_i \leq \pi_{Stop}^{\mathcal{R}} = \lambda^{\mathcal{R}}$ and similarly $\pi_j^{\mathcal{R}} + s' < \lambda^{\mathcal{R}}$, $-\lambda^{\mathcal{R}} < \pi_i^{\mathcal{R}} + s - \pi_j^{\mathcal{R}} - s' < \lambda^{\mathcal{R}}$. Hence, the equality (2) gives:

$$\pi_i^{\mathcal{R}} + s = \pi_j^{\mathcal{R}} + s' \text{ and } \mathcal{R}(T_i) + q = \mathcal{R}(T_j) + q'$$

Hence, T_i and T_j are performed on the same time slot $\pi_i^{\mathcal{R}} + s$ in the acyclic schedule $\pi^{\mathcal{R}}$. Thus, $T_j \in F_{\pi_i^{\mathcal{R}} + s}$ and then $F_t \subseteq F_{s + \pi_i^{\mathcal{R}}}$

Since $\pi^{\mathcal{R}}$ fulfills the resource constraints induced by $G^{\mathcal{R}}$, $F_{\pi_i^{\mathcal{R}}}$ (and then F_t) meets the resource constraints.

For precedence constraints, we need to prove that, $\forall (T_i, T_j) \in E, \forall q \in \mathbb{N}$:

$$\begin{aligned} \sigma^{\mathcal{R}}(T_i, q) + p_i + l_{ij} &\leq \sigma^{\mathcal{R}}(T_j, q + h_{ij}) \\ \Leftrightarrow \pi_i^{\mathcal{R}} + (q + \mathcal{R}(T_i)) \lambda^{\mathcal{R}} + p_i + l_{ij} &\leq \pi_j^{\mathcal{R}} + (q + \mathcal{R}(T_j) + h_{ij}) \lambda^{\mathcal{R}} \end{aligned}$$

Hence, we have to verify that inequality (3) is satisfied for each (T_i, T_j) in E and for any $q \in \mathbb{N}$

$$\pi_i^{\mathcal{R}} - \pi_j^{\mathcal{R}} + p_i + l_{ij} \leq (\mathcal{R}(T_j) - \mathcal{R}(T_i) + h_{ij}) \lambda^{\mathcal{R}}. \quad (3)$$

Case 1 : $(T_i, T_j) \in G^{\mathcal{R}}$

Then, $\mathcal{R}(T_j) - \mathcal{R}(T_i) + h_{ij} = 0$. Since $\pi^{\mathcal{R}}$ fullfills the precedence constraints induced by $G^{\mathcal{R}}$,

$$\pi_i^{\mathcal{R}} + p_i + l_{ij} \leq \pi_j^{\mathcal{R}}.$$

Hence, inequality 3 is satisfied.

Case 2 : $(T_i, T_j) \notin G^{\mathcal{R}}$

Thus, $\mathcal{R}(T_j) - \mathcal{R}(T_i) + h_{ij} > 0$. And then, using 1

$$\pi_i^{\mathcal{R}} - \pi_j^{\mathcal{R}} + p_i + l_{ij} \leq \pi_i^{\mathcal{R}} + p_i + l_{i,Stop} + l_{j,Start} - \pi_j^{\mathcal{R}}$$

As $\pi_j^{\mathcal{R}} \geq l_{j,Start}$ and $\pi_i^{\mathcal{R}} + p_i + l_{i,Stop} \leq \pi_{Stop}^{\mathcal{R}}$, we get:

$$\begin{aligned} \pi_i^{\mathcal{R}} - \pi_j^{\mathcal{R}} + p_i + l_{ij} &\leq \pi_{Stop}^{\mathcal{R}} \\ &\leq \lambda^{\mathcal{R}} \\ &\leq \lambda^{\mathcal{R}}(\mathcal{R}(T_j) - \mathcal{R}(T_i) + h_{ij}) \end{aligned}$$

which achieves the proof. ■

Now, the idea, previously used by [8] and [10] is to choose a particular retiming and use a guaranteed algorithm to get a schedule $\pi^{\mathcal{R}}$ of $G^{\mathcal{R}}$, and then to extend the guarantee to the induced periodic schedule.

List scheduling algorithms are the most used heuristics for scheduling with precedence and resource constraints. Alix *et al.* [11] prove that such algorithms have the following worst case performance in the case of m identical processors in presence of precedence delays:

$$C_{max} \leq \left(2 - \frac{1}{m(\rho + 1)}\right) C_{max}^{opt}$$

where $\rho = \frac{l_{max}}{p_{min}}$. For systems with unit execution time tasks and typed processors, Chou and Chung [12] give the following bound:

$$C_{max} \leq \left(k + 1 - \frac{1}{m_x(l_{max} + 1)}\right) C_{max}^{opt}$$

We thus propose the following generic algorithm 1 to solve our problem, by using a list algorithm to produce $\pi^{\mathcal{R}}$.

Algorithm 1: Extended DSP

1. Find a legal retiming $\mathcal{R}$ for G ;
 2. **for** $(T_i, T_j) \in E$ **do**
 - if** $\mathcal{R}(T_j) - \mathcal{R}(T_i) + h_{ij} = 0$ **then**
 - keep** (T_i, T_j) in $G^{\mathcal{R}}$; add nodes Start and Stop;
 3. Perform a list scheduling on $G^{\mathcal{R}}$ coping with both precedence and resource constraints. Compute $\pi_i^{\mathcal{R}}$ the start time of task T_i in this schedule and $\lambda^{\mathcal{R}} = C_{max}(G^{\mathcal{R}}) = \pi_{Stop}^{\mathcal{R}}$;
 4. Define the cyclic schedule $\sigma^{\mathcal{R}}$ by:
 - for** $1 \leq q \leq N$ **do**
 - for** $T_i \in V$ **do**
 - keep** $\sigma^{\mathcal{R}}(T_i, q) = \pi_i^{\mathcal{R}} + \lambda^{\mathcal{R}}(q + \mathcal{R}(T_i))$;
-

The acyclic graph provided by the retiming $\mathcal{R}$ is given by Figure 2 and its corresponding list scheduling allocation is presented in Figure 3. The makespan of this

pattern is $\lambda^{\mathcal{R}} = 9$ and it gives the period of the modulo scheduling of G . The different steps of this heuristic are illustrated by algorithm 1.

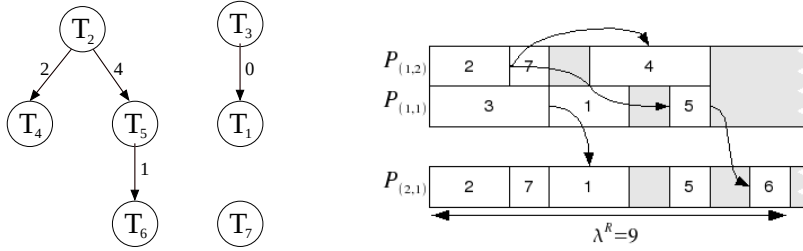


Fig. 2 The acyclic graph **Fig. 3** A pattern generated by a list scheduling of $G^{\mathcal{R}}$: $\lambda^{\mathcal{R}} = 9$. given by the retiming $\mathcal{R}$.

2 Worst case analysis

In this section we analyze the worst case performance of algorithm 1 in general, making use of the proof of Chou and Chung [12] for list scheduling. Then, we show that using some particular retiming, that can be computed in polynomial time, we can get an overall guarantee for the Extended DSP algorithm.

2.1 Minimal length of pattern

Consider a dependence graph G . An acyclic graph $G^{\mathcal{R}}$ is obtained by a retiming $\mathcal{R}$. Then, we schedule $G^{\mathcal{R}}$ by a list algorithm and generate a pattern $\pi^{\mathcal{R}}$. We note $\phi^{\mathcal{R}}$ the length (sum of the delays and processing times) of the longest path in $G^{\mathcal{R}}$. Let λ^{opt} be the optimal period of G .

We consider two types of bounds obtained from resource and precedence constraints.

2.1.1 Resource bound

Lemma 2 For each type r , let m_r be the number of machines of type r . Then,

$$\lambda^{opt} \geq \max_{1 \leq r \leq k} \frac{\sum_{i=1}^n b_r^i \cdot p_i}{m_r}.$$

Proof. The shortest time required to complete the tasks using resources of type r on a single machine is $\sum_{i=1}^n b_r^i \cdot p_i$. Hence, on m_r parallel processors, the shortest time required is $\frac{\sum_{i=1}^n b_r^i \cdot p_i}{m_r}$. Furthermore, the length of the optimal period λ^{opt} is not shorter than the time required to schedule these tasks once. ■

2.1.2 Precedence bounds

Let $\pi^{\mathcal{R}}$ be a schedule induced by a list algorithm on $G^{\mathcal{R}}$. In order to reveal the dependencies among tasks, we classify the time slots into three kinds:

1. A full slot t_f is a time slot in which at least all the processors of a certain type are executing tasks.
2. A partial slot t_p is a time slot in which at least one processor of each type is idle and this slot contains at least one non-idle processor.
3. A delay slot t_d is a time slot in which all processors are idle.

We note:

- p : the number of partial slots in $\pi^{\mathcal{R}}$.
- d : the number of delay slots in $\pi^{\mathcal{R}}$.

Lemma 3 *The partial-slots lemma:*

If $\pi^{\mathcal{R}}$ contains p partial slots and d delay slots, then $\phi^{\mathcal{R}} \geq p + d$.

Proof. We prove this lemma by finding a chain $h = \langle T_{j_1}, \dots, T_{j_c} \rangle$ in $\pi^{\mathcal{R}}$ such that the length of h is at least equal to $p + d$.

Let $T_{j_c} = \text{Stop}$ and assume that we already have a chain $\langle T_{j_{i+1}}, \dots, T_{j_c} \rangle$. Consider, if it exists, the predecessor T_{j_i} of $T_{j_{i+1}}$ such that $\pi_{j_i}^{\mathcal{R}} + p_{j_i} + l_{j_i, j_{i+1}}$ is maximum.

The construction of h leads to the following observation: All the slots before $\pi_{j_i}^{\mathcal{R}}$ or between $\pi_{j_i}^{\mathcal{R}} + p_{j_i} + l_{j_i, j_{i+1}}$ and $\pi_{j_{i+1}}^{\mathcal{R}}$ (if they exist) are full slots and in which there is no available processor in a type of resource used by $T_{j_{i+1}}$. Otherwise, $T_{j_{i+1}}$ would have been scheduled earlier by the list algorithm.

Therefore, all the p partial slots and d delay slots are covered by the intervals $[\pi_{j_i}^{\mathcal{R}}, \pi_{j_i}^{\mathcal{R}} + p_{j_i} + l_{j_i, j_{i+1}})$, so that the length of h is not less than $p + d$. Thus, $\phi^{\mathcal{R}} \geq p + d$. ■

Lemma 4 *The delay-slots lemma:*

If $\pi^{\mathcal{R}}$ contains d delay slots, then $\phi^{\mathcal{R}} \geq d + \left\lceil \frac{d}{l_{max}} \right\rceil p_{min}$.

Proof. The schedule is computed by a list algorithm, then the number of any consecutive delay slots is not greater than l_{max} . Consider the chain h defined in the previous lemma. All the delay slots are included in $\cup_{1 \leq i \leq c-1} [\pi_{j_i}^{\mathcal{R}} + p_{j_i}, \pi_{j_i}^{\mathcal{R}} + p_{j_i} + l_{j_i, j_{i+1}})$, since during interval $[\pi_{j_i}^{\mathcal{R}}, \pi_{j_i}^{\mathcal{R}} + p_{j_i})$, T_{j_i} is performed. Now the length of each interval $[\pi_{j_i}^{\mathcal{R}} + p_{j_i}, \pi_{j_i}^{\mathcal{R}} + p_{j_i} + l_{j_i, j_{i+1}})$ is less than l_{max} . So it holds that $c \cdot l_{max} \geq d$. The length of h is thus not less than d plus the length of the chained tasks, which is greater than $c \cdot p_{min} \geq \left\lceil \frac{d}{l_{max}} \right\rceil p_{min}$. Thus, $\phi^{\mathcal{R}} \geq d + \left\lceil \frac{d}{l_{max}} \right\rceil p_{min}$. ■

2.2 Performance bound

Here we define the notations to be used below:

$$M = \sum_{1 \leq r \leq k} m_r.$$

u_r : the number of non-idle cycles on processors of type r .

v_r : the number of non-idle cycles on processors of type r in partial slots in $\pi^{\mathcal{R}}$.
 $V = \sum_{1 \leq r \leq k} v_r$: the number of non-idle cycles in partial slots in $\pi^{\mathcal{R}}$.

We now prove a first bound on the algorithm performance based on the length of the longest path in $G^{\mathcal{R}}$.

Theorem 1 Consider a dependence graph G . Let $\mathcal{R}$ be a legal retiming $\mathcal{R}$ on G and $\phi^{\mathcal{R}}$ the length of the longest path in $G^{\mathcal{R}}$. Then,

$$\frac{\lambda^{\mathcal{R}}}{\lambda^{opt}} \leq k + \left(1 - \frac{1}{m_x(\rho + 1)}\right) \frac{\phi^{\mathcal{R}}}{\lambda^{opt}}.$$

Proof. Consider the pattern $\pi^{\mathcal{R}}$:

$$M\lambda^{\mathcal{R}} = \text{number of non-idle cycles} + \text{number of idle cycles.}$$

where the second term on the right hand side can be decomposed as the number of idle cycles occurring during delay, partial and full slots.

1. The number of idle cycles per processor occurring during delay slots is equal to Md .
2. The number of idle cycles per processor occurring during partial slots is at most equal to $Mp - V$.
3. Using Lemma 2, we now give a bound on the number of idle cycles occurring during full slots.

Notice that for a resource type r , there are at most $\frac{u_r - v_r}{m_r}$ full slots in which resource r is full. Thus the number of idle cycles in these slots is at most $(M - m_r) \frac{u_r - v_r}{m_r}$. Hence we get:

$$\begin{aligned} &\leq \sum_{1 \leq r \leq k} (M - m_r) \frac{u_r - v_r}{m_r} \\ &\leq \sum_{1 \leq r \leq k} (M - m_r) \frac{u_r}{m_r} - \sum_{1 \leq r \leq k} (M - m_r) \frac{v_r}{m_r} \\ &\leq \sum_{1 \leq r \leq k} (M - m_r) \max_{1 \leq i \leq k} \frac{\sum_{j=1}^n b_r^i \cdot p_i}{m_r} - (M - m_x) \frac{\sum_{1 \leq r \leq k} v_i}{m_x} \\ &\leq (kM - M)\lambda^{opt} - (M - m_x) \frac{V}{m_x} \end{aligned}$$

Then,

$$\begin{aligned} M\lambda^{\mathcal{R}} &\leq M\lambda^{opt} + (kM - M)\lambda^{opt} - (M - m_x) \frac{V}{m_x} + Mp - V + Md \\ &\leq kM\lambda^{opt} - (M - m_x) \frac{V}{m_x} + Mp - V + Md \\ &\leq kM\lambda^{opt} - M \frac{V}{m_x} + Mp + Md \end{aligned}$$

V is the number of non-idle cycles in partial slots, since a partial slot contains at least one non-idle cycle, $V \geq p$. Thus,

$$\begin{aligned}
\lambda^\sigma &\leq k\lambda^{opt} - \frac{p}{m_x} + p + d \\
&\leq k\lambda^{opt} + (1 - \frac{1}{m_x})(p + d) + \frac{1}{m_x}d \\
&\leq k\lambda^{opt} + (1 - \frac{1}{m_x})(p + d) + \frac{1}{m_x}d \left(1 + \frac{p_{min}}{l_{max}}\right) \left(\frac{1}{1 + \frac{p_{min}}{l_{max}}}\right) \\
&\leq k\lambda^{opt} + (1 - \frac{1}{m_x})(p + d) + \frac{1}{m_x} \left(d + \left\lceil \frac{d}{l_{max}} \right\rceil p_{min}\right) \left(\frac{\rho}{\rho + 1}\right)
\end{aligned}$$

From Lemmas 3 and 4, we have $\phi^{\mathcal{R}} \geq p + d$ and $\phi^{\mathcal{R}} \geq d + \left\lceil \frac{d}{l_{max}} \right\rceil p_{min}$. Then,

$$\begin{aligned}
\lambda^{\mathcal{R}} &\leq k\lambda^{opt} + (1 - \frac{1}{m_x})\phi^{\mathcal{R}} + \frac{1}{m_x} \frac{\rho}{\rho + 1} \phi^{\mathcal{R}} \\
&\leq k\lambda^{opt} + (1 - \frac{1}{m_x(\rho + 1)})\phi^{\mathcal{R}}
\end{aligned}$$

■

2.3 Choosing a good retiming

In order to improve the performance bound, it seems important to minimize the ratio between $\phi^{\mathcal{R}}$ and λ^{opt} . So if we have a good lower bound LB of λ^{opt} , using theorem 7 in [13], we can verify the existence of a legal retiming $\mathcal{R}'$ such that $\phi^{\mathcal{R}'} \leq LB \leq \lambda^{opt}$. If such retiming exists, we have a performance guarantee of:

$$\frac{\lambda^{\mathcal{R}'}}{\lambda^{opt}} \leq k + 1 - \frac{1}{m_x(\rho + 1)}.$$

An another approach consists on minimizing the length of the longest path in the pattern. There are well-known retiming algorithms [13] to minimize $\phi^{\mathcal{R}}$. Let $\mathcal{R}_{opt}$ be a retiming for which the length of the longest path in the acyclic graph $G^{\mathcal{R}_{opt}}$ is minimal. We note it ϕ^{opt} . We also denote by σ_∞ an optimal periodic schedule for unlimited resources (with period λ_∞).

Lemma 5 Let $\delta = \max_{(T_i, T_j) \in E} (p_i + l_{ij})$ be the maximum scope in G . Then,

$$\lambda_\infty + \delta - 1 \geq \phi^{opt}.$$

Proof. Consider an optimal cyclic schedule σ_∞ for unlimited resources. Let us define $r : V \rightarrow [0, \lambda_\infty - 1]$ and $\mathcal{R} : V \rightarrow \mathbb{Z}$ such that:

$$\sigma_\infty(T_i, q) = r(T_i) + \lambda_\infty (q + \mathcal{R}(T_i)), \quad \forall T_i \in V, \forall q \in \mathbb{N}$$

Then, the precedence constraint for each edge $(T_i, T_j) \in E$ is:

$$\begin{aligned}
\sigma_\infty(T_i, q) + p_i + l_{ij} &\leq \sigma_\infty(T_j, q + h_{ij}) \\
r(T_i) + \lambda_\infty (q + \mathcal{R}(T_i)) + p_i + l_{ij} &\leq r(T_j) + \lambda_\infty (q + h_{ij} + \mathcal{R}(T_j)) \\
r(T_i) + p_i + l_{ij} &\leq r(T_j) + \lambda_\infty (h_{ij} + \mathcal{R}(T_j) - \mathcal{R}(T_i))
\end{aligned}$$

[10] proved that $\mathcal{R}$ defines a valid retiming for G . Furthermore, $G^{\mathcal{R}}$ is obtained by keeping the edges of G for which $\mathcal{R}(T_j) + h_{ij} - \mathcal{R}(T_i) = 0$. Thus ,

$$r(T_i) + p_i + l_{ij} \leq r(T_j), \quad \forall (T_i, T_j) \in E$$

Let $h = \langle T_{j_1}, \dots, T_{j_c}, Stop \rangle$ be a chain in $G^{\mathcal{R}}$.

$$r(T_{j_i}) + p_{j_i} + l_{j_i, j_{i+1}} \leq r(t_{j_{i+1}}), \forall i \in \{1, \dots, c-1\},$$

By summing up these $c-1$ inequalities, we have

$$r(T_{j_1}) + \sum_{i=1}^{c-1} (p_{j_i} + l_{j_i, j_{i+1}}) \leq r(T_{j_c})$$

Thus, $\sum_{i=1}^{c-1} (p_{j_i} + l_{j_i, j_{i+1}}) \leq r(T_{j_c})$. This inequality is true for any chain of $G^{\mathcal{R}}$ in particular for the longest path in $G^{\mathcal{R}}$. Hence,

$$\underbrace{\sum_{i=1}^{c-1} (p_{j_i} + l_{j_i, j_{i+1}}) + (p_{j_c} + l_{j_c, Stop})}_{\phi^{\mathcal{R}}} - \underbrace{(p_{j_c} + l_{j_c, Stop})}_{\leq \delta} \leq \underbrace{r(T_{j_c})}_{\leq \lambda_{\infty} - 1}$$

Hence, $\phi^{\mathcal{R}} - \delta \leq \lambda_{\infty} - 1$ and since $\phi^{opt} \leq \phi^{\mathcal{R}}$, we have the desired result. ■

Finally, from lemma5, we deduce our last performance ratio.

Theorem 2 Consider a dependence graph G . Let $\mathcal{R}_{opt}$ be a retiming on G that minimize the length of the longest path in $G^{\mathcal{R}}$. Then,

$$\lambda^{\mathcal{R}_{opt}} \leq \left(k + 1 - \frac{1}{m_x(\rho + 1)}\right) \lambda^{opt} + \left(1 - \frac{1}{m_x(\rho + 1)}\right) (\delta - 1).$$

Proof. Using theorem 1 applied to $\mathcal{R}_{opt}$, we get:

$$\begin{aligned} \lambda^{\mathcal{R}_{opt}} &\leq k\lambda^{opt} + \left(1 - \frac{1}{m_x(\rho + 1)}\right) \phi^{opt} \\ &\leq k\lambda^{opt} + \left(1 - \frac{1}{m_x(\rho + 1)}\right) (\lambda^{\infty} + \delta - 1) \\ &\leq k\lambda^{opt} + \left(1 - \frac{1}{m_x(\rho + 1)}\right) (\lambda^{opt} + \delta - 1) \\ &\leq \left(k + 1 - \frac{1}{m_x(\rho + 1)}\right) \lambda^{opt} + \left(1 - \frac{1}{m_x(\rho + 1)}\right) (\delta - 1) \end{aligned}$$

■

3 Conclusion

Instruction scheduling, which takes place when compiling applications for modern processors, affects critically the performance of the the overall system cost and energy consumption.

In this paper, we presented a generalized model of instruction scheduling but our results are still available for applications in production systems problems.

We have built upon results of [8, 10, 12] and extended them to propose a guaranteed heuristic for unitary resource-constrained modulo scheduling problems. A worst case analysis of this heuristic is explored and a performance bound is established. It is the first guarantee derived for cyclic scheduling problems in the case of many different resources.

We point out that it would be interesting to derive algorithms more sophisticated than list scheduling to improve this performance bound.

Finally, it would be worth to study the importance of choosing a good retiming and its impact on the performance guarantee. Good lower bounds for the optimal period would give a better guarantee.

References

1. M., P.J., X., X.: Modélisation, analyse et optimisation des systèmes à fonctionnement cyclique. Masson (1995)
2. Hanen, C., Munier, A.: Cyclic scheduling on parallel processors: An overview. In: P. Chrétienne, E.G. Coffman, J.K. Lenstra, Z. Liu (eds.) *Scheduling theory and its applications*. J. Wiley and sons (1994)
3. Rau, B.R.: Dynamically scheduled vliw processors. In: MICRO 26: Proceedings of the 26th annual international symposium on Microarchitecture, pp. 80–92. IEEE Computer Society Press, Los Alamitos, CA, USA (1993)
4. Allan, V.H., Jones, R.B., Lee, R.M., Allan, S.J.: Software pipelining. *ACM Comput. Surv.* **27**(3), 367–432 (1995)
5. Rau, B.R.: Iterative modulo scheduling: an algorithm for software pipelining loops. In: MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture, pp. 63–74. ACM, New York, NY, USA (1994)
6. Artigues, C., Dinechin, B.D.: Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. C. Artigues, S. Demassey, E. Nron (2008)
7. Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research* **112**(1), 3–41 (1999)
8. Gasperoni, F., Schwiegelshohn, U.: Generating close to optimum loop schedules on parallel processors. *Parallel Processing Letters* **4**, 391–403 (1994)
9. Wang, J., Eisenbeis, C., Jourdan, M., Su, B.: Decomposed software pipelining: a new perspective and a new approach. *Int. J. Parallel Program.* **22**(3), 351–373 (1994). DOI <http://dx.doi.org/10.1007/BF02577737>
10. Calland, P.Y., Darte, A., Robert, Y.: Circuit retiming applied to decomposed software pipelining. *IEEE Trans. Parallel Distrib. Syst.* **9**(1), 24–35 (1998). DOI <http://dx.doi.org/10.1109/71.655240>
11. Munier, A., Queyranne, M., Schulz, A.S.: Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In: IPCO, pp. 367–382 (1998)
12. Chou, H.C., Chung, C.P.: Upper bound analysis of scheduling arbitrary delay instruction on typed pipelined processors. *Int. Journal of High Speed Computing* **4**(4), 301–312 (1992)
13. Leiserson, C.E., Saxe, J.B.: Retiming synchronous circuitry. NASA STI/Recon Technical Report N **89**, 17,797–+ (1988)

Scheduling policy for on-line translation service

Malgorzata Sterna · Jacek Juraszek

Abstract The research concerns the scheduling tool supporting the web service offering automatic text translation between Polish and three other languages. The web service accepts client orders submitted via WWW site and realizes them using specialized translation software. The system consists of a few translation processes dedicated to particular translation directions, which run in parallel sharing hardware resources. After the deep analysis of real historical data, obtained for a simple random scheduler, we proposed an optimized on-line scheduling policy for managing client orders, based on a priority dispatching rule. It improved the efficiency of the web service, fulfilling all the requirements specified by the owner of the system.

1 Introduction

In the last decade the number of users of web services has increased significantly, causing problems in ensuring the satisfactory quality of these on-line services. Applying even very simple scheduling techniques may be very helpful in improving the efficiency of such systems. In this paper, we investigate the web service for automatic language translation, provided by one of the Polish publishing companies. It offers six translation directions between a national language and three foreign languages: English, German and Russian. The translation process is performed by the highly specialized software designed by academic experts. It enables translating not only single words but whole sentences too, taking into account grammatical rules of particular languages. The service is provided free of charge via WWW site. Users can submit requests of the length restricted by the owner of the system to 999 characters. Moreover, it is assumed that translation tasks have to be completed within the 180-second time limit. After this deadline a task is canceled and a user receives the information about the failure in the translation process. The owner of the system assumed that at most 0.5% of client orders

Malgorzata Sterna
Institute of Computing Science, Poznan University of Technology
E-mail: malgorzata.sterna@cs.put.poznan.pl

Jacek Juraszek
Institute of Computing Science, Poznan University of Technology
E-mail: jacek.juraszek@cs.put.poznan.pl

may be non-served for any reason. All user requests submitted via WWW site are managed by a control procedure, which separates them to different translation processes based on the required translation direction. Two most popular translation directions from/to English are served by three identical processes each, while the remaining ones are associated with only one process. Hence, the web service is realized by 10 translation processes, which are theoretically independent. However, since these processes share system resources (Intel Xeon X5355 2.66 GHz computer with 8 processors and 7GB RAM) they influence each other. In practice, the processing speed depends on the workload of the web service. During the rush hours, when the number of submitted requests increases significantly, a translation task may require more time due to processors time sharing. Thus, the processing time for a particular translation task is unknown a priori and it depends on the contents of the user request and, additionally, on many technical issues such as e.g.: the system workload, runtime errors in the translation software or communication delays. Hence, we deal with the on-line scheduling problem, where user orders are not known in advance and they appear dynamically in the system, whose state changes in the non-deterministic way.

Originally, the control algorithm managing user tasks was based on simple First-In-First-Out (FIFO) rule (cf. e.g. [1], [6]) and random assignment of English tasks to translation processes (for the remaining types of tasks only one process is available). However the quality of service was unsatisfactory due to too many non-served client requests. The percentage of non-completed tasks exceeded the 0.5% limit established by the owner of the service. Within our research, we proposed an optimized control procedure increasing the quality of the analyzed web service in the extent meeting all system requirements. The performance of the new scheduling policy was checked in extended computational experiments, using the historical long-period data collected in the web service log.

The organization of the paper is as follows. Section 2 presents a formal definition of the problem under consideration. Section 3 shows the analysis of real values of the web service parameters, which allowed us to design a new efficient scheduling policy adjusted to the specificity of the system. The idea of this on-line algorithm is presented in Section 4. Section 5 contains the main observations resulting from the computational experiments. The paper is concluded in Section 6.

2 Problem Formulation

The web service under consideration can be modeled as a specific on-line case of the multi-purpose machine scheduling problem (cf. e.g. [2], [3]). At a certain time moment, all user requests for translating texts submitted to the web service form the set of available tasks $T = \{T_1, \dots, T_n\}$, while ten translation processes running within the system can be modeled as the set of multi-purpose machines $M = \{M_1, \dots, M_{10}\}$. Obviously, particular six translation directions offered by the web service are associated with given subsets of machines (translation processes). Hence, translations from and to English are served by any machine from set $E_1 = \{M_1, M_2, M_3\}$ and $E_2 = \{M_4, M_5, M_6\}$, respectively. Translations from/to German, can be realized by only one machine, i.e. $G_1 = \{M_7\}$ and $G_2 = \{M_8\}$, respectively. Similarly, user requests concerning translations from/to Russian are executed by only one process, i.e. $R_1 = \{M_9\}$ and $R_2 = \{M_{10}\}$, respectively. The parameters of a task become known only after submitting a user request to the web service. After the task arrival to the

system it is available for processing. Particular tasks T_j , i.e. requests for text translation, are described by the following parameters (assuming that time parameters are given in seconds which passed from the last restart of the system):

- the arrival (release) time: r_j ,
- the deadline: d_j ($d_j = r_j + 180$),
- the number of characters in the translated text: c_j ($c_j \leq 999$),
- the number of words in the translated text: w_j .

Moreover, depending on the translation direction required by a user, we define for task T_j the subset of machines $\mu_j \in \{E_1, E_2, G_1, G_2, R_1, R_2\}$, on which it can be executed.

In general, each task T_j has to be performed by one machine selected from set μ_j without preemption. However the machines (i.e. translation processes) work in the non-deterministic way and our knowledge about their state is very restricted. Actually, the machines model the processes associated with the specialized translation software, designed by external developers, which still contains some implementation errors. Unfortunately, for unknown reasons, not all tasks finish with success and, due to runtime errors, translation processes may need restarting by the operating system. Consequently, machines (modeling these processes) may be unavailable in some periods, but these unavailability periods cannot be predicted and even detected. To overcome this difficulty, the processing time limit for a single attempt to task execution was introduced. After exceeding this limit, the execution of a task is considered as unsuccessful. If task T_j is not completed successfully, then its execution is repeated possibly on another machine from set μ_j . The failure in execution may result from: exceeding the time limit, the unavailability of a machine or an internal software error in the translation process. Hence, in general, tasks cannot be preempted, but there might be a few attempts to their completion in the system. This means that the execution of a task can be preempted only due to the technical reasons mentioned before. These multiple attempts to completing a task can be considered as non-resumable preemptions of tasks (cf. e.g. [4], [7]).

From the theoretical point of view, particular translation directions (sets of machines $E_1, E_2, G_1, G_2, R_1, R_2$) might have been considered as separate scheduling problems with 3 parallel machines for English (E_1, E_2) and with a single machine for the remaining two languages (G_1, G_2, R_1, R_2), because sets μ_j defined for particular types of tasks (i.e. translation directions) are disjoint. However, we decided to model the web service as the one integral system to reflect the goal of the research, which is to improve the quality of the whole web service. Moreover, as we have mentioned, although particular translation processes, represented by machines, are independent they share hardware resources and, in practice, influence one another. This influence cannot be expressed formally, but it may be observed in the system log data. The huge number of tasks requesting one machine consumes processors time and may slow down other machines. Runtime errors occurring in one translation process may make the system unstable and lead to the failure of other processes. The machines in the formal model correspond to software processes not to physical processors, which those processes share. Of course, it is possible to apply different scheduling strategies for particular types of tasks, but the performance of the proposed policies should be validated with regard to the entire system.

As we have mentioned, the owner of the system imposed two efficiency constraints, which should be obeyed by a scheduling policy: the 180-second time limit for particular client requests and at most 0.5% of requests non-completed by the service within this time limit. Consequently, the main goal of any policy controlling the web service is to

execute as many tasks as possible and to ensure short response time of the system. To evaluate the quality of the service, we collect a few task parameters, which become available after processing task T_j by the web service, namely:

- the completion time: C_j ,
- the flow time (the system response time): $F_j = C_j - r_j$,
- the number of trials of task execution: a_j ,
- the processing time in trial i : $p_j(i), i = 1..a_j$,
- the machine used in trial i : $M_j(i) \in \mu_j, i = 1..a_j$,
- the status of trial i : $s_j(i) \in \{S, T, F, M\}, i = 1..a_j$.

If the last trial of executing task T_j finished successfully ($s_j(a_j) = S$), then the processing time $p_j(a_j)$ denotes the translation time for task T_j . The unsuccessful execution of a task may result from exceeding the time limit ($s_j(i) = T$), from the failure in translation ($s_j(i) = F$) or from the machine failure, i.e. the runtime error of the translation software ($s_j(i) = M$). The processing time consumed by the unsuccessful trials can be considered as wasted one, since these trials did not lead to the task completion. Obviously, such trials cannot be usually avoided, because they result from different kinds of failures, which appear in the real Internet service. However a scheduling policy should detect them as early as possible, minimizing the total wasted time, which is an additional parameter reflecting the efficiency of the system, besides the percentage of non-completed tasks and the average response time mentioned above.

3 Web Service Analysis

The translation service under consideration was originally managed by a very simple scheduling policy. It assigned user requests submitted to the system (tasks) to a proper translation process (machine) in a random way (in the case of a few processes/machines which may serve a task). The queues associated with particular translation processes (machines) were handled according to the First-In-First-Out (FIFO) strategy. This simple approach resulted in too low quality of service in terms of the average system response time and, first of all, of the percentage of non-completed user requests.

In order to optimize the scheduling policy, we carefully analyzed the web service log to determine the possible ways of improving the algorithm. This analysis was restricted to a single working day for which one of peak workloads was detected. The selected data set contained 60750 translation requests, i.e. nearly one request submitted every second. Actually, the number of tasks handled by the web service is rather stable over the week, so restricting the analysis to one day is fully justified. The number of tasks present in the system changes significantly during the day (cf. Figure 1). The rush hours are between 15:00 and 19:00, when the number of tasks is nearly 30 times higher than for the hour with the minimum load (6:00-7:00). The numbers of requests submitted for particular translation directions differ very much too. The majority of submitted requests concerns translation from English to Polish (47.91%) and from Polish to English (35.38%), i.e. together 83% of all submissions. These two translation directions are handled by three identical processes each (three identical parallel machines). Requests involving German amount to about 11% (5.8% of German-Polish translations and 5.31% in the opposite direction), while submissions involving Russian amount to only 6% of the total number of tasks (3.7% of Polish-Russian translations and 1.89% in the opposite direction). These translation directions are served by only one process each (single machines). The analysis of the web service log suggested two

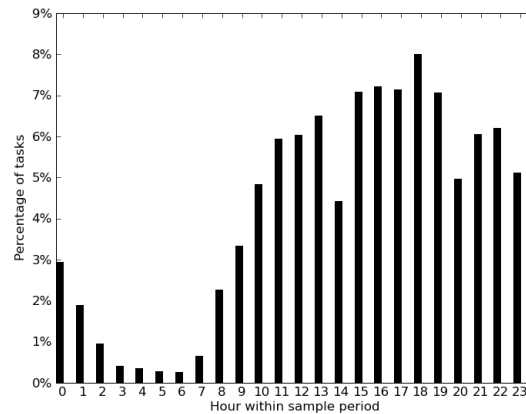


Fig. 1 The number of tasks in particular subperiods of the 24-hour sample with 60750 tasks

directions of improving the scheduling policy. Due to the heavy workload of parallel machines associated with English translations, the more efficient way of assigning tasks to machines is required (than the random one). Moreover the historical data showed that the system encountered quite numerous runtime errors of translation software and failures in text translations. For example, some user requests could not be served due to grammatical mistakes within the submitted text. Such tasks locked a translation process (a machine) until exceeding a given time limit (deadline), although they could not have been successfully completed at all. Unfortunately, the scheduler cannot distinguish translations in progress (tasks which will be successfully completed in future) from translations which encounter internal software errors (unsuccessful trials of task executions). This limitation of the web service suggested another way of improving the scheduling policy: tuning the time limit after which a particular trial of a task execution is considered as unsuccessful and after which this attempt should be aborted.

As it was mentioned, one of the ways of optimizing the scheduling policy is assigning tasks to machines according to a certain priority dispatching rule. Most commonly used rules are based on the processing times of tasks (cf. e.g. [1], [6]), which are unknown in the analyzed problem. In order to predict the value of this task parameter, we looked for the correlation between the processing time, i.e. the translation time known after completing a task, and the task description known before its execution. From the scheduler point of view, without any domain-specific knowledge, the translation tasks can be characterized only by the number of words and the number of characters in the submitted text. Unfortunately, no correlation has been observed between these two parameters and the task processing time for the whole system (0.003 and -0.008 for the number of words and characters, respectively) as well as for particular translation directions (cf. Table 1, which shows the correlation coefficient for the number of words in translated text, the results for the number of characters were similar). The processing time includes not only the pure translation time, but also the time necessary for managing a task, which is independent of the text length. It seems that the translated texts were too short (due to the limit imposed by the owner of the web service) to show the influence of the text length on the processing time and the task processing time

Table 1 The correlation coefficient for the task processing time and the number of words

Translation direction	Correlation coefficient	Average processing time [sec.]	Std. dev. of processing time [sec.]	Average words number	Std. dev. of words number
pl→en	-0.051	4.7	2.4	8	15
en→pl	-0.059	7.7	6.3	16	28
pl→ge	-0.004	6.3	6.2	8	13
ge→pl	0.402	5.2	3.4	12	21
pl→ru	-0.212	3.9	1.5	9	23
ru→pl	-0.384	3.1	1.7	17	26

depends more on the state of the runtime environment than on the length of the text. In consequence, the optimized scheduling policy should rather focus on enforcing multiple trials of executing a task, hoping for success, than on scheduling tasks according to their predicted processing times, waiting for their completion in the first attempt.

4 Scheduling Policies

The web service under consideration is built from a few software components designed by different companies. For this reason, the scheduler does not have the full control of the system and it has to take into account the specificity of the translation processes. It accepts user requests and executes tasks associated with them on particular machines (i.e. translation processes). The scheduler manages the queues associated with machines deciding on the order of task executions. Moreover, it supervises a particular task execution, handling possible errors or terminating the executions which exceeded the given time limit. Taking into account two facts that: the task processing time is unknown and it depends rather on the system workload than on the submitted text, as well as that: all tasks have to be completed within the same 180-second limit, the queues associated with machines should be scheduled according to FIFO strategy (i.e. according to task arrival times r_j). This priority dispatching rule was applied in both scheduling policies: the simple one, used originally in the web service, and in the optimized one, proposed within this research. In consequence, the only way to increase the efficiency of the system was optimizing the algorithm handling the execution of a task. The general idea of this method is presented in Algorithm 1.

First the algorithm validates the deadline for task T_j , i.e. it checks whether less than 180 seconds passed since submitting it to the system at time r_j . This constraint is checked at every attempt to completing task T_j . Particular trials are numbered with i from 1 to a_j . Variable t represents the current time in the system counted in seconds from the last restart of the web service. If the deadline for a task is exceeded, its execution fails due to timeout ($s_j(i) = T$) and the user receives the information on the failure in text translation. Otherwise the scheduler tries to execute task T_j . First, procedure $ChooseMachine(T_j, \mu_j, i)$ determines machine $M_j(i)$ selected from a set of eligible machines μ_j which will be used for processing T_j in trial i . Then procedure $Translate(T_j, i, M_j(i), p_j^{max}(i))$ puts T_j in the queue for $M_j(i)$ and supervises its execution, guaranteeing that the deadline ($r_j + 180$) will not be exceeded. Moreover, it assures that the execution time $p_j(i)$ will not be longer than the time limit given for the i -th attempt to the task completion on machine $M_j(i)$, i.e. than $p_j^{max}(i)$. If the i -th attempt finishes successfully ($s_j(i) = S$), then the translated text is sent to the user.

Algorithm 1 Serving Translation Request (T_j)

```

 $i \leftarrow 1$ 
loop
  if  $t \geq r_j + 180$  then
     $s_j(i) \leftarrow T$ 
    break
  end if
   $M_j(i) \leftarrow \text{ChooseMachine}(T_j, \mu_j, i)$ 
   $(s_j(i), p_j(i)) \leftarrow \text{Translate}(T_j, i, M_j(i), p_j^{max}(i))$ 
  if  $s_j(i) = S$  or  $t \geq r_j + 180$  then
    break
  end if
   $i \leftarrow i + 1$ 
end loop
 $a_j \leftarrow i$ 

```

Otherwise, the scheduler encounters the failure in the task completion and it tries to complete it once more in the next $(i + 1)$ -st attempt.

The simple scheduling policy performed the random selection of a machine from set μ_j and applied the same processing time limit $p_j^{max}(i) = 180$ in every trial and for all machines. The simple strategy assumed that machines are reliable and tasks are likely to be completed at the first attempt. The historical data showed that these assumptions were unrealistic. The optimized policy assigns task T_j to the machine with the shortest queue of tasks waiting for execution (if $|\mu_j| > 1$). In the case of failure in task execution, at the next $(i + 1)$ -st attempt to completing T_j , the scheduler eliminates from the analysis machine $M_j(i)$, which was not able to perform the task in the previous i -th trial (in the case of one more trial this machine will be available again). The analysis of the web service log showed that failures often result from errors in the translation process, which needs restarting by the operating system. Within this restarting time the process (machine) is non-available, but the scheduler is not informed about the machine state. Shifting tasks to another machine from μ_j makes its successful execution more likely, because the translation process represented by another machine may be normally working (not being restarted) and may be executed by another processor with lower workload.

In order to detect the machine non-availability, the optimized scheduling policy applies the much shorter time limit for a single attempt to task execution than the simple strategy. Moreover, this time limit is not constant, but it increases with the trial number. In this way, the scheduling policy allows for translating some specific texts, which are linguistically correct, but require the longer processing time than on average. Additionally, the time limit is determined separately for particular machines (processes) in the system, based on the data collected in the web service log. Consequently, the time limit is not common for the whole web service, but it has been tuned for particular translation directions separately, reflecting their efficiency, reliability, error resistance etc. Summing up, the processing time limit for the i -th attempt to completing task T_j on machine $M_j(i)$ is determined as: $p_j^{max}(i) = \min\{P95\%_{M_j(i)} + (i-1) * SD_{M_j(i)}, 180 - \sum_{k=1}^{i-1} p_j(k)\}$, where $P95\%_{M_j(i)}$ denotes 95-th percentile of the processing time for successful translations on machine $M_j(i)$ (i.e. the upper bound of the processing time for 95% of tasks executed on this machine based on the web service log), $SD_{M_j(i)}$ denotes the standard deviation of the processing time for successful translations on

machine $M_j(i)$. The second factor corresponds to the time limit decreased with the durations of previous trials for this task (if any).

5 Computational Experiments

The optimized scheduling policy was compared to the simple scheduler in the computational experiments performed for real data extracted from the web service log described in Section 3. The system log was used as input data for the new algorithm. Hence, both strategies were evaluated for the same set of 60750 tasks submitted by the users of the web service within 24-hour period. Both policies received the same stream of user requests in terms of their contents and arrival times. For each task its execution history was logged including its final status (determined by the result returned by the translation algorithm or by the decision of the scheduler).

The optimized policy increased significantly the efficiency of the service and met all requirements defined for the system. The simple strategy resulted in 0.57% of non-completed tasks. This level was unacceptable due to the 0.5%-limit imposed by the owner of the web service. The optimized algorithm decreased the percentage of unsuccessful translations more than 3 times to only 0.18%. However, these values, aggregated for the whole 24-hour period, do not reflect the dynamics of the web service. The percentage of faults in task executions in particular subperiods (cf. Figure 2) showed that even at rush hours the optimized policy did not exceed the 0.5% limit. For example, in period 15:00-16:00, it led to less than 0.2% of faults, while the simple algorithm failed in executing nearly 2% of tasks. For the heavy workload of the web service, the system efficiency depends strongly on the scheduling policy, because the queues associated with particular machines become longer and exceeding the deadline for particular tasks is more likely. In such case, the execution of tasks must be optimized - the simple strategy is definitely not sufficient. When the number of tasks submitted to the system is smaller, then both algorithms behave similarly, since faults in tasks completions result mainly from the translation failures, which cannot be eliminated - they will appear for any scheduling policy, because of e.g. grammatical mistakes in the submitted texts.

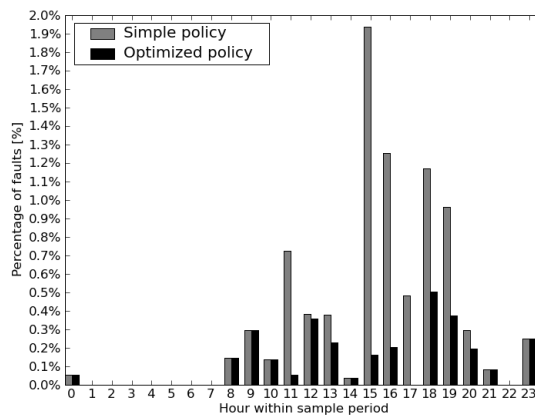


Fig. 2 Percentage of non-executed tasks in particular subperiods from 60750 tasks

Table 2 The efficiency factors for particular translation directions for both policies

Translation direction	Percentage of faults [%]		Average flow time [sec.]	
	simple policy	optimized policy	simple policy	optimized policy
pl→en	0.00	0.00	5.94	3.79
en→pl	0.83	0.03	12.46	4.04
pl→ge	3.07	3.04	8.06	5.85
ge→pl	0.00	0.00	6.29	5.24
pl→ru	0.18	0.18	4.24	4.04
ru→pl	0.00	0.00	3.23	3.20

This observation is confirmed by the analysis of the percentage of faults encountered for particular translation directions (cf. Table 2). The software modules designed for particular translation directions differ in their quality. The high percentage of faults in translations from Polish to German resulted from the low quality of the software. The number of tasks associated with this service was relatively small (about 5%), therefore the system could have executed all of them without exceeding time limit. The failures in execution were caused by translation errors, independent of the scheduling policy. For this reason, the improvement observed for the optimized strategy was insignificant. But for the service translating from English to Polish, for which the number of submitted tasks was the largest one, the percentage of faults decreased from 0.83% to 0.03%. In this case, the quality of translating software was high and faults in task executions resulted mainly from exceeding the deadline.

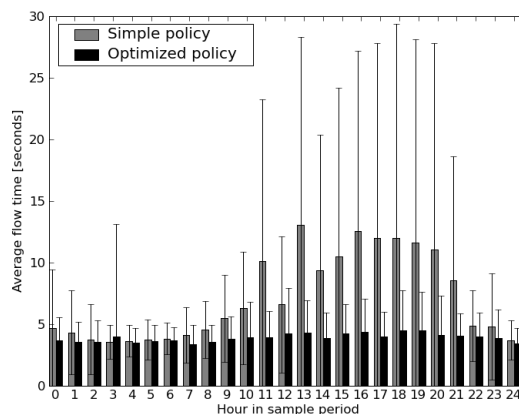


Fig. 3 The average task flow time in particular subperiods with the standard deviation values

The optimized scheduling policy not only decreased the number of non-completed tasks, but it also shortened the response time of the system from 9.1 to 4.1 seconds on average, i.e. the new strategy reduced the time the users waited for text translation more than twice. Moreover, the system behavior became very stable. The analysis of the average task flow time (i.e. the system response time F_j) in particular subperiods (cf. Figure 3) disclosed the weakness of the simple policy. The response time grew with the system workload. In rush hours, users had to wait for text translation nearly three

times as long as in the night. Furthermore the response time was unpredictable, which is reflected in the huge standard deviation values. The optimized policy ensured the stable response time independently of the system workload. The flow time reduction (and the stabilization of its value) is also visible for particular translation directions (cf. Table 2), especially for the most occupied service translating from English to Polish. For this service the process of scheduling tasks became crucial due to the large number of user requests present in the system at the same time.

The improvement of the quality of the web service, visible in the decreased percentage of non-executed tasks and the shortened average response time, resulted mainly from changing the scheduling policy by: imposing simple load-balancing among parallel machines and enforcing multiple trials of task executions in case of failures. These changes in the strategy influenced also other parameters describing the efficiency of the system. They caused the reduction in the average processing time of tasks (from 6.2 to 3.6 seconds) and the time spent by the web service on unsuccessful text translations (from 7.29% to 4.44% of the total CPU time). Moreover, the new strategy eliminated the failures resulting from the system errors caused by overloading it and decreased the number of failures resulting from exceeding the time limit.

6 Conclusions

The paper presented the on-line algorithm for scheduling tasks in the real web service for text translation. The efficiency of the existing simple strategy was not sufficient, since it did not meet the requirements specified by the owner of the system. The Internet service was formally described as the complex scheduling environment with two separate sets of parallel identical machines and four single machines, corresponding to different translation directions. Although tasks submitted for particular types of translations are scheduled separately, they share the hardware resources and they had to be analyzed simultaneously. The schedule has to be constructed on-line in real-time, since the set of tasks is unknown in advance and unexpected events appear in the system. We proposed the list scheduling algorithm, which handles tasks submitted to the web service and fully collaborates with the specialized translation software.

The computational experiments showed that the optimized policy increased the efficiency of the web service. It fulfilled all requirements specified by the system owner, which had not been satisfied by the simple policy used so far. Thus, the main goal of the research was reached. The optimized strategy replaced the previous one in the web service. Moreover, the analysis of test results showed some weak points of the system, independent of the scheduling policy, such as the low quality of some modules of translation software, as well as the lack of the mechanism detecting failures in translation processes.

Due to the increasing number of the web service users the system under consideration has been recently enlarged by changing its architecture and adding additional hardware and software resources. Within the future research we will have to propose a new formal model of the web service and to design scheduling policies adjusted to its specificity. Moreover, although the algorithms running in the web service are deterministic (the differences in the processing times of particular attempts for a task execution result from the dynamic character of the environment in which these algorithms are executed not from the randomness of their decisions) we would like to take into account the results obtained in the literature for randomized approaches (cf. e.g. [5]).

References

1. J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz, Handbook on scheduling. From theory to applications, Springer. Berlin-Heidelberg-New York (2007)
2. P. Brucker, Scheduling algorithms, Springer. Berlin-Heidelberg-New York (2007)
3. P. Brucker, B. Jurisch, A. Kreamer, Complexity of scheduling problems with multi-purpose machines, *Annals of Operations Research* 70, 57–73 (1997)
4. C.-Y. Lee, Machine scheduling with availability constraints, in: J.Y.-T. Leung (Ed.), *Handbook of scheduling: algorithms, models and performance analysis*, 22.1– 22.13, Chapman & Hall/CRC. London (2004)
5. M. Luby, A. Sinclair, D. Zuckerman, Optimal speedup of Las Vegas algorithms, *Information Processing Letters* 47, 173–180, (1993)
6. M. Pinedo, *Scheduling: theory, algorithms and systems*, Springer. New York (2008)
7. G. Schmidt, Scheduling with limited machine availability, *European Journal of Operational Research* 121, 1–15, (2000)

A Nonlinear Mixed Integer Program for Search Path Planning Problem (SPPP)

Adel Guitouni · Hatem Masri

Abstract In this paper, we propose a nonlinear integer program to model a path planning for a single airborne search asset in a continuous space and time for a fixed time period, through a connected space. The intent is to maximize the detection of a cooperative target (e.g., search and rescue). The proposed model is based on the assumption of existing a priori information (e.g., result of information fusion) to establish a spatial distribution of possible locations. Solution of the nonlinear program provides a path as well as effort allocation to each location. We illustrate the results of the paper on an empirical example.

Key words: Search and Rescue problem, Vehicle Routing Problem, Nonlinear Integer Programming.

1 Introduction

In this paper, we consider the search path problem for a single airborne search asset in a continuous space and time for a fixed time period T , through a connected region with the intent of maximizing the detection of a cooperative target (search and rescue). *A priori* information (e.g., sensors data, experts opinion) is available to build a spatial distribution of the possibility of locating the target. Such distribution might be a probability, possibility, fuzzy or any other type of uncertainty modelling. A seaker or a searcher is an airborne platform equipped with sensors (e.g., scanning radar, imaging radar, electro-optic, infra-red). For simplicity of the model, we assume the probability of detecting the target is function of the time spent searching in a particular geographic

Adel Guitouni

1. Defense R&D Canada, 2459 Pie XI Noth, Quebec (Quebec), G3J 1X5, Canada

2. Faculty of Business, University of Victoria, Victoria (British Columbia), Canada

3. Faculté des sciences de l'administration, Université Laval, Quebec (Quebec), Canada

E-mail: Adel.Guitouni@drdc-rddc.gc.ca

Hatem Masri

LARODEC laboratory, Institut Suprieur de Gestion

41 rue de la libert, Le Bardo 2000

University of Tunis, Tunisia

E-mail: hatem.masri@isg.rnu.tn

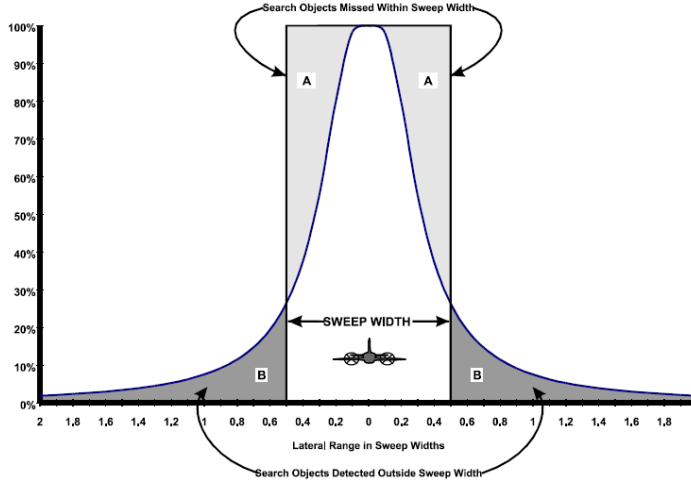


Fig. 1 Sweep Width [18]

region. We simplify the conditional probability to be expressed in term of the lateral range function of the sensor, its sweep width W , and the total time spent in the region.

We assume that the probability of detection is given by the following expression:

$$POD = 1 - e^{-\left(\frac{W_j \cdot v_j}{A_j}\right)t_j} \quad (1)$$

where W_j is the effective sweep width in region j , A_j is the surface of region j and v_j is the platform speed in region j . In this paper we propose a path planning algorithm to maximize the probability of finding the cooperative target within a given time window. This paper is organized as follows. In section 2, we present an overview of the search path planning problem. We review related work in section 3. We propose the model in section 4. We discuss the results and present the conclusions in section 5.

2 Search Path Planning Problem

Resource management for conducting search and rescue, surveillance and reconnaissance mission is a very important problem for many organizations. This problem is characterized by the employment of mobile (e.g. maritime patrol aircraft, helicopters, UAVs, ships) and fixed surveillance assets (e.g. land radar) to a large geographic area in order to identify, assess and track the maximum number of moving, stopped or drifting objects. The observed objects are not necessarily aware of being observed and are cooperative or non-cooperative, and friendly or hostile. The scarce surveillance (e.g. Electro-Optical (EO), Infra-Red (IR), and Synthetic Aperture Radar (SAR) sensors), tracking capabilities (normal radar modes), and search and rescue assets makes it very difficult to problem due to the time and space constraints. A search mission is usually planned for a given search area while minimizing cost in term of time, distance and exposure to threats, and maximizing the chances of success subject to a set of constraints imposed by the amount of effort available for the search.



Fig. 2 Example of Search Path for Multiple Platforms

The search path planning problem is intractable and complex because it is multi-faceted and NP-hard even for stationary targets [7]. Many elements should be considered including:

- the platform configurations and properties which generally related to the physical and functional properties of the platform (aircraft),
- the number of platforms and their configurations,
- the sensors, or the instruments used to carry out the seeker such as the eye, radar, sonar, television, and cameras,
- the object or the target being searched which may be stationary or moving, cooperative or non cooperative, friendly or hostile),
- the geographic or physical search area that may be continuous (in Euclidean n-space) or discrete (a set of cells), closed or open, and
- the available search effort may be continuous (i.e. measured by time or track length) or discrete (i.e. measured by a finite number of scans or looks).

Let a seeker s be a combination of a given platform with appropriate sensors for a given search situation. Multiple seekers might be assigned different search areas. Search path in each area might follow a static or dynamic patterns as shown in Fig. 2. This paper address optimal path planning given a particular seeker. We focus on airborne seekers.

We assume the process shown in Fig. 3. A distress call might be received by a search and rescue center, which will activate its prioritization process. Information is received from different information sources (e.g., sensors). An information fusion process produce a spatial distribution of possible location of the target. Through a filtering process, one can generate a synthesized distribution function over the search area.

We assume that *a priori* information is available through information fusion processes about a plausible distribution that represent the possibility that the target is

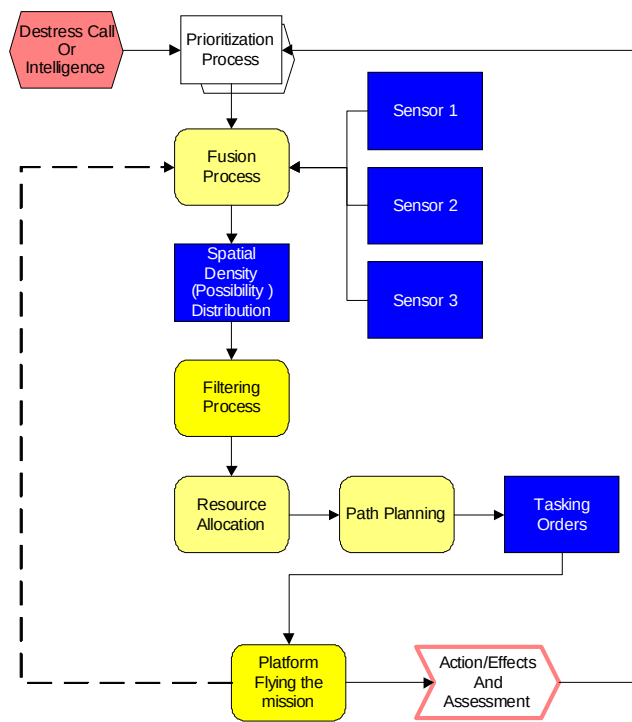


Fig. 3 Simplified Process

within a particular containment area (geographic location as shown by Fig. 4). The intensity of the distribution might refer to a higher degree of confidence (e.g., probability) that the object might be in a particular more confined region of that containment area. At this point, the distribution function is not necessarily a probability distribution function. Different clustering approaches might be applied in order to create a more intelligible distribution map. Now, the problem is to fly an airborne seeker with the intent to find the target while optimizing a set of objectives under constraints.

3 Related work

The search-path problem has been closely related to search effort allocation problem [11]. Search theory perspective [8] supposes that targets and seekers are modelled probabilistically. Search theory was developed for the first time by Koopman and the U.S. Navy's operations research group [12] in order to develop better strategies for anti-submarine warfare during the Second World War. Since then, search theory was mainly applied in application fields such as: surveillance, SAR missions and exploration (e.g. Finding the US Nuclear Submarine Scorpion lost in 1968 and Clearing unexploded ordnance in the Suez Canal). A "good" search plan should maximize the Probability of Success (*POS*), which is the probability of finding the search object. To compute the *POS*, two other probabilities are used. The Probability of Containment (*POC*), which is the probability that the search object is in a particular cell of the searched area, and

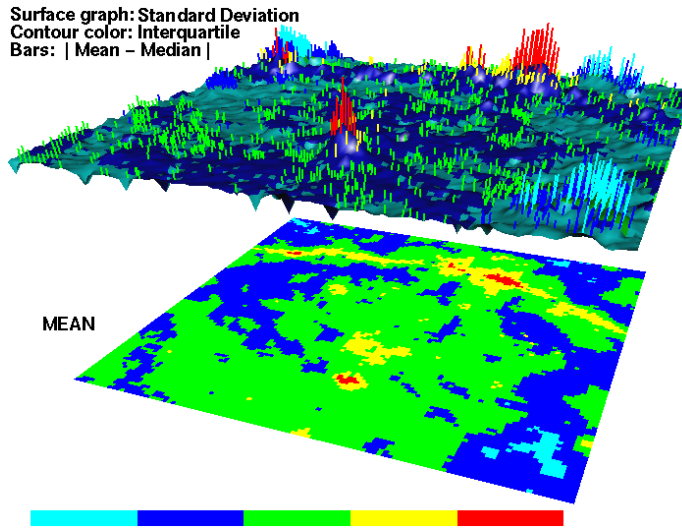


Fig. 4 Example distribution over a surface [10]

the Probability of Detection (POD) which is the probability that the sensor detects the search object given that it is in the area searched. Note that the probability POD is function of the effort provided by the sensor on a specific area. The effort may be measured by time, area searched, track length or any other appropriate measure. The relationship between these three probabilities is as follows:

$$POS = POC \times POD \quad (2)$$

The first step of this process consists in identifying the boundaries of the possibility area (PA) that may contain the search object. Subjective (e.g., rules, experts judgements) and objective (e.g., historical data, fused sensors data) knowledge are generally used to define the limits of this area. For instance, the International Aeronautical and Maritime Search and Rescue (IAMSAR) Manual [9] contains some guidelines to establish the possibility area. Other approaches like possibility distribution function or fuzzy functions might be used. Then, the possibility area is discretized into J cells indexed by $j = 1 \dots J$, which represent the smallest regions over which search effort can be allocated. A search within each cell might follow a predefined or dynamic pattern.

The discrete path problem is based on the assumption that the searcher or seeker and the target are roaming in a discrete time and space dimensions. According to Kierstead and DelBalzo [11] "virtually all work on this problem has been under the the additional assumptions that: (a) detection can only occur when the searcher and target are in the same cell; and (b) detection opportunities are independent across time steps". Moreover, discrete-time moving-target search model usually assumes the Markovian property of the target's movement [13]. Dambreville and Le Cadre [14] proposed an algorithm for optimal search for a target following a Markovian movement or a conditionally deterministic motion when search resources are renewed with generalized linear constraints.

The optimal search path problem is NP-Hard even if the target is stationary [15]. An optimal dynamic programming method is proposed by [19]. Stewart [20] has proposed

a search efforts relocation heuristic whenre the effort is restricted to paths. Eagle and Yee [22] proposed an extension of the work of Stewart's [20]. They proposed an optimal branch-and-bound algorithm limited 7X7 search grids. Other extensions of this work might be found in [23] and [24].

Hong et al. [13] proposed a pseudo-polynomial heuristic for the single-searcher path-constrained discrete-time Markovian-target search. The heuristic is based on an approximate of non detection probability computed from the conditional probability that reflects the search history over a fixed time windows. Jacobson et al. [16] proposed simultaneous generalized hill climbing algorithms (SGHC) to determine optimal search strategies over multiple search platforms for the MTSP. Kierstead and DelBalzo [11] proposed a genetic algorithm (GA) for planning search paths against a moving target. In his paper, Janez [17] proposed to model the sensors planning problem as vehicle routing problem (VRP). The formulation is based on the assumption that the coordinate of the targets to be visited as known a priori. Unlike previously mentioned works, targets are stationary and their location is well defined. The problem addressed is then the allocation of different surveillance assets to visiting each target. This is a classical VRP problem.

4 Model formulation

In this work we take a different approach to the search path problem. First The Search and Rescue problem using an only one search engine ($SARP_1$) presents some similarity compared to the distance constrained vehicle routing problem (DVRP). The DVRP is to find a set of vehicle tours that begins and ends at the start vertex, traverses a subset of the set of vertices and must have a total duration not exceeding a given limit. Each node is visited exactly once a time. This last constraint makes the difference between the $SARP_1$ problem and the DVRP. The similarity between the two problems is that they look for a tour that starts from a given vertex. The difference is that in the DVRP problem, we are obliged to visit all vertices which is not the case with $SARP_1$. The $SARP_1$ is equivalent to find a tour (or set of tours), that maximizes the sum of the probability of success of visited regions under time restriction.

4.1 Graph definition

Let's assume the search problem shown in Fig. 5. An airborne platform should fly over a geographic area flowing a given path to maximize the chances to find the target. The aircraft has a limited flying time and the target might have a limited survival time. The different countours represent different level of possible location with more or less higher chances for the position of the target. The concentration of the countours represent higher possibilities. One can apply a cut thresholds and obtain the distribution of Fig. 6. Therefore, the airborne platform might fly over the different region and spend more time where the countours are more dense and less elsewhere as shown in Fig. 7. Such flying path might be represented by Fig. 8.

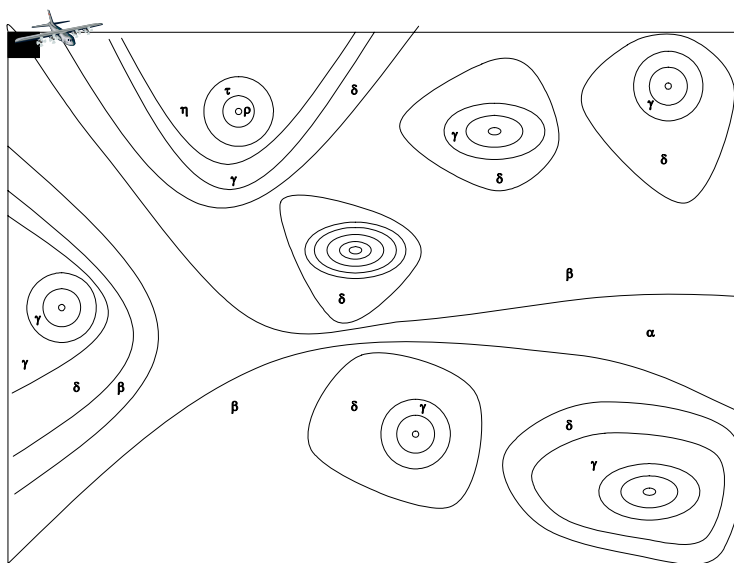


Fig. 5 2D view of a distribution of possible locations of the target

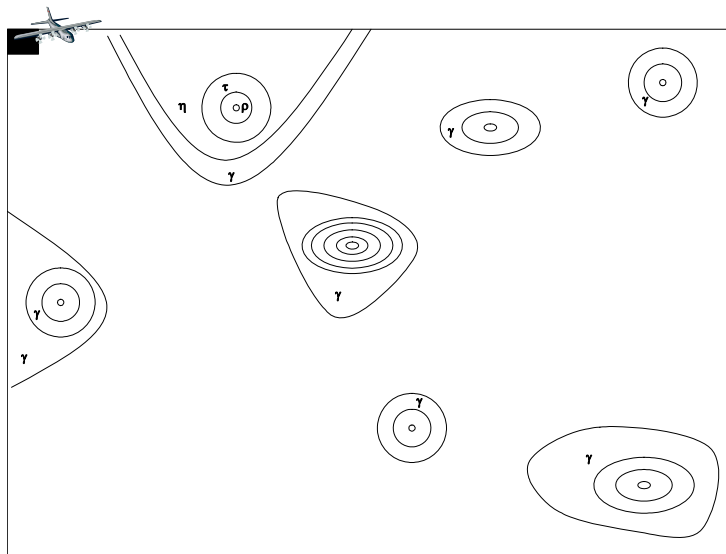


Fig. 6 δ Cut

According to the previous section, we end here with a graph $G = (V, E)$ with n vertices. We suppose that the search engine starts at the vertex $i = 1$. Poc_i is the probability of containment for each vertex i , ($Poc_1 = 0$). To travel between two vertices i and j , the search engine takes t_{ij} time ($t_{ii} = 0$ for all i).

We note that, we consider only vertices that verify $\{i \mid t_{1i} + t_{i1} \leq T_M\}$ where T_M is the time window for the search and rescue mission.

4.2.1 Variables

Let us denote by

- x_{ij} a binary variables. If $x_{ij} = 1$ then the search engine after visiting the region i will over fly the region j . Otherwise, such a travel between region i and j will not be considered in the optimal flight plan.
- t_i the time of search over the region i .

In the following, we propose a set of constraints and objectives to model the SARP₁ problem.

4.2.2 The constraints

The first constraints restricts the time of search to the given time of mission T_M

$$\sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n (t_{ij} + t_j) x_{ij} \leq T_M \quad (3)$$

The second set of constraints ensures the flow conservation in vertices:

$$\sum_{\substack{i=1 \\ i \neq j}}^n x_{ij} - \sum_{\substack{s=1 \\ s \neq j}}^n x_{js} = 0, j = 1, \dots, n \quad (4)$$

The third constraint ensures that any tour must include the initial vertex 1

$$\sum_{i=2}^n x_{i1} = 1 \quad (5)$$

The fourth set of constraints restricts the optimal tour to visit at most one time each vertex

$$\sum_{\substack{i=1 \\ i \neq j}}^n x_{ij} \leq 1, j = 2, \dots, n \quad (6)$$

The last set of constraints must be set to eliminate subtours. In the literature, many formulations were proposed for the subtours elimination problem [3]. Among them the Miller-Tucker-Zemlin (MTZ) formulation [2]:

$$\begin{aligned} u_j - u_i &\geq 1 + p(x_{ij} - 1), i \neq j, i = 2, \dots, n, j = 2, \dots, n \\ 1 &\leq u_j \leq n - 1, j = 2, \dots, n \end{aligned} \quad (7)$$

where $u_i, i = 2, \dots, n$ are positive variables and p is the number of visited vertices in the tour. For the case $p = n$, the MTZ formulation may be used for the travel salesman problem. Many extensions and lifting of the MTZ formulation were proposed in the literature essentially for the travel salesman problem [4]. One of these extensions of the MTZ formulation was provided by Desrochers and Laporte [6] to the vehicle routing problem:

$$\begin{aligned} u_j - u_i &\geq c_{ij} + M(x_{ij} - 1), i \neq j, i = 2, \dots, n, j = 2, \dots, n \\ s_j + (c_{1j} - s_j) x_{1j} &\leq u_j \leq L - r_j - (c_{j1} - r_j) x_{j1}, j = 2, \dots, n \end{aligned} \quad (8)$$

where c_{ij} is the distance associated to the edge (i, j) , s_j the length of a shortest path from vertex 1 to vertex j , r_j is the length of a shortest path from vertex j to vertex 1, L is the maximum tour length and $M \geq \max_{i,j} \{L - s_j - r_i + c_{ij}\}$. Under the following correspondences: $L = T_M$, $c_{ij} = t_{ij} + r_j$, $s_j = c_{1j}$ and $t_j = c_{j1}$, the Desrochers and Laporte formulation (8) became as follows

$$\begin{aligned} u_j - u_i &\geq t_j + t_{ij} + M(x_{ij} - 1), i \neq j, i = 2, \dots, n, j = 2, \dots, n \\ t_{1j} + t_j &\leq u_j \leq T_M - t_{j1}, j = 2, \dots, n \end{aligned} \quad (9)$$

where $M \geq \max_{i,j,i \neq j} \{T_M - t_{1i} + t_{ij} + t_j - t_{j1}\}$. The set of constraints (9) is not valid for the SARP₁ as we suppose that $u_j \geq t_{1j} + t_j$ for all $j \geq 2$ which will oblige the optimal tours to include all vertices. This lower bound is only valid if the vertex j is visited by optimal tours (i.e. $\sum_{i=1, i \neq j}^n x_{ij} = 1$). Therefore, we propose to reformulate the the

Desrochers and Laporte formulation as follows

$$\begin{aligned} u_j - u_i &\geq t_j + t_{ij} + M(x_{ij} - 1), i \neq j, i = 2, \dots, n, j = 2, \dots, n \\ (t_{1j} + t_j) \sum_{i=1, i \neq j}^n x_{ij} &\leq u_j, j = 2, \dots, n \\ u_j &\leq T_M - t_{j1}, j = 2, \dots, n \end{aligned} \quad (10)$$

where $M \geq \max_{i,j,i \neq j} \{T_M - t_{1i} + t_{ij} + t_j\}$.

Proposition 1 *The formulation (3) is valid for the SARP₁.*

Proof To prove that the formulation (3) is valid for SARP₁, we have to verify that constraints in (3) eliminate subtours and that the length of feasible subtour do not exceed mission time T_M .

Consider a subtour that does not include the vertex 1: $l_1, \dots, l_p$ where $l_k \neq 1$ for $k = 1, \dots, p$ and $l_p = l_1$. Therefore, $x_{l_k l_{k+1}} = 1$, $k = 1, \dots, p-1$ and summing from 1 to $p-1$ the constraints $u_{l_{k+1}} - u_{l_k} \geq t_{l_{k+1}} + t_{l_k l_{k+1}}$ yields to $\sum_{k=1}^{p-1} (t_{l_{k+1}} + t_{l_k l_{k+1}}) \leq 0$ which is a contradiction and then all tours must include the vertex 1.

We are going to prove that any subtour length does not exceed the time of mission T_M . Consider the following subtour: $1, l_1, \dots, l_p, 1$ where $l_k \neq 1$ for $k = 1, \dots, p$ and L_s the length of this subtour. Summing up the constraints $u_{l_{k+1}} - u_{l_k} \geq t_{l_{k+1}} + t_{l_k l_{k+1}}$, $k = 1, \dots, p-1$ gives

$$\sum_{k=1}^{p-1} (t_{l_{k+1}} + t_{l_k l_{k+1}}) \leq u_{l_p} - u_{l_1} \quad (11)$$

Since $t_{1l_k} - t_{l_k} \leq u_{l_k} \leq T_M - t_{l_k 1}$ for $k = 1, \dots, p$ then

$$\begin{aligned} L_s &= t_{1l_1} + t_{l_1} + t_{l_p 1} + \sum_{k=1}^{p-1} (t_{l_{k+1}} + t_{l_k l_{k+1}}) \\ &\leq t_{1l_1} + t_{l_1} + t_{l_p 1} + u_{l_p} - u_{l_1} \\ &\leq T_M \end{aligned} \quad (12)$$

4.2.3 Objective function

We have to define Pos_j

The objective function is to maximize the sum of the probability of success of the visited (vertices) region

$$\sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n Poc_j \left(1 - e^{-\left(\frac{w_j v_j}{A_j}\right) t_j} \right) x_{ij} \quad (13)$$

4.2.4 The nonlinear mixed integer programming Model

The following expression provides the nonlinear mixed interger program:

$$\begin{aligned} Max \quad & \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n Poc_j \left(1 - e^{-\left(\frac{w_j v_j}{A_j}\right) t_j} \right) x_{ij} \\ s.t. \quad & \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n (t_{ij} + t_j) x_{ij} \leq T_M \\ & \sum_{\substack{i=1 \\ i \neq j}}^n x_{ij} - \sum_{\substack{s=1 \\ s \neq j}}^n x_{js} = 0, j = 1, \dots, n \\ & \sum_{i=2}^n x_{i1} = 1 \\ & \sum_{\substack{i=1 \\ i \neq j}}^n x_{ij} \leq 1, j = 2, \dots, n \\ & t_j \leq T_M \times \sum_{\substack{i=1 \\ i \neq j}}^n x_{ij}, j = 2, \dots, n \\ & u_j - u_i \geq t_j + t_{ij} + M(x_{ij} - 1), i \neq j, i = 2, \dots, n, j = 2, \dots, n \\ & u_j \geq (t_{1j} + t_j) \times \sum_{\substack{i=1 \\ i \neq j}}^n x_{ij}, j = 2, \dots, n \\ & u_j \leq (T_M - t_{j1}) \times \sum_{\substack{i=1 \\ i \neq j}}^n x_{ij}, j = 2, \dots, n \\ & x_{ij} \in \{0, 1\}, i = 1, \dots, n, j = 1, \dots, n, i \neq j \\ & u_j \geq 0, j = 2, \dots, n \\ & t_j \geq 0, j = 1, \dots, n \end{aligned} \quad (14a)$$

where $M \geq \max_{i,j,i \neq j} \{T_M - t_{i1} + t_{ij} + t_j\}$.

5 Computational Study

Let's consider an example of 10 regions to visit based on the filtering of the spatial distribution function. The following table shows the travel time between each couple of regions. Region 1 represent the departure and destination of the platform.

t_{ij}	1	2	3	4	5	6	7	8	9	10
1	0	45	60	70	65	100	50	42	28	58
2	45	0	14	12	15	11	10	8	14	15
3	60	14	0	11	15	18	16	90	11	12
4	70	12	11	0	11	18	14	16	12	11
5	65	15	15	11	0	15	15	14	15	10
6	100	11	18	18	15	0	13	15	11	8
7	50	10	16	14	15	13	0	99	9	9
8	45	8	90	16	14	15	99	0	48	7
9	28	14	11	12	15	11	9	48	0	14
10	58	15	12	11	10	8	9	7	14	0

Let $KA = \left(\frac{W_j v_j}{A_j} \right)$. The following table provides an empirical example for the value of KA and the probability of containment in each region to be visited. KA is assumed the same for each region.

	1	2	3	4	5	6	7	8	9	10
KA	$2.e^{-4}$	$2.e^{-4}$	$2.e^{-4}$	$2.e^{-4}$	$2.e^{-4}$	$2.e^{-4}$	$2.e^{-4}$	$2.e^{-4}$	$2.e^{-4}$	$2.e^{-4}$
POC	0	0.1	0.15	0.15	0.05	0.1	0.11	0.1	0.17	0.07

This problem is then formulated and solved the problem using Lingo 9.0 and we obtained the following results:

- Decision Variables: 121 with 90 integers
- Number of constraints: 123 with 84 nonlinear
- Runtime: 53 seconds
- Objective: 0.053
- Iterations: 20770
- Optimal path: 1-9-3-4-2-1

The solution is as follows:

	1	2	3	4	5	6	7	8	9	10
Visit Time	0	0	389,04	389,04	0	0	0	0	1014,84	0

6 Discussion and Conclusion

The proposed model is a simplification of the real problem. The model optimize the path as well as the effort to be allocated to each region to be searched. To our knowledge, the model offers an original approach to a particular case of a surveillance problem. However, it offers a better approach to path planning. The formulation of the problem is extendable to consider other constraints and other type of objective functions. A 10 vertices problem has been solved with commercial solver within 50 seconds.

A heuristic might be used to bring down this time. This model will be extended by adding additional heterogenous platforms. Other constraints and objectives might be considered.

The proposed model assumes a static state of the world. The path is generated prior to flying the mission and no additional information is gained over time. But, information is gained over time. The fusion process will then update the spatical distribution, which leads to a dynamic search problem.

References

1. Dantzig, G., D. Fulkerson, S. Johnson. 1954. Solution of a large scale traveling salesman problem. *Oper Res.* 2 393-410.
2. Miller, C., A. Tucker, R. Zemlin. 1960. Integer programming formulation of traveling salesman problems. *J. ACM* 7 326-329.
3. A. J. Orman and H. P. Williams, A Survey of Different integer programming formulations of the travelling salesman problem, Research report 04.67 Department of Operational Research, London School of Economics and Political Science, 2004.
4. SHERALI, H.D. and DRISCOLL, P.J., "On tightening the relaxations of Miller-Tucker-Zemlin formulations for asymmetric traveling salesman problem", *Operations Research* 50 (2002).
5. Kara I, Laporte G and Bekta A (2004). A note on the lifted Miller-Tucker-Zemlin subtour elimination constraints for the capacitated vehicle routing problem. *Eur J Opl Res* 158: 793-795
6. M. Desrochers, G. Laporte, Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints, *Operations Research Letters* 10 (1991) 27-36.
7. Trummel, J.R. and Weisinger, J.R., The Complexity of the Optimal Searcher Path Problem, *Operations Research* 34 (1986) 324-327.
8. Stone, L.D., *Theory of Optimal Search*, Academic Press (1975).
9. IAMSAR, International Aeronautical and Maritime Search And Rescue Manual, ICAO/IMO publications, (1999).
10. Kao, D., Dungan, J., & Pang, A., Visualizing 2D Probability Distributions from EOS Satellite Image-Derived Data Sets: A Case Study. *Proc. of IEEE Visualization*, San Diego, USA (2001) 457-460.
11. Kierstead, D. P. and D. R. DelBalzo, A genetic algorithm applied to planning search paths in complicated environments, *Military Operations Research* 8(2): 45-59 (2003).
12. Koopman, B. O., *Search and Screening: General Principles with Historical Applications*, Pergamon Press (1980).
13. Hong, S.-P., S.-J. Cho, et al., A pseudo-polynomial heuristic for path-constrained discrete-time Markovian-target search, *European Journal of Operational Research* 193(2): 351-364 (2009).
14. Dambreville, F. and J.P. Le Cadre, Detection of a Markovian-target with optimization of the search efforts under generalized linear constraints, *Naval Research Logistics* 49 (2002) 117-142.
15. K.E. Trummel, J.R. Weisinger, The complexity of the optimal searcher path problem, *Operations Research* 34 (1986) 324-327.
16. Jacobson, S. H., L. A. McLay, et al., Optimal search strategies using simultaneous generalized hill climbing algorithms, *Mathematical and Computer Modelling* 43(9-10): 1061-1073 (2006).
17. Janez, F., Optimization method for sensor planning, *Aerospace Science and Technology* 11(4): 310-316 (2007).
18. Frost, J., *The Theory of Search A Simplified Explanation*, Report by Soza & Company Ltd. and Office of Search and Rescue US Coast Guard, (1998).
19. Eagle, J.N., The optimal search for a moving target when the search path is constrained, *Operations Research* 32 (1984) 1107-1115.
20. Stewart, T.J., Search for a moving target when searcher motion is restricted, *Computers and Operations Research* 6 (1979) 129-140.

21. Brown, S. S., Optimal search for a moving target in discrete time and space, *Operations Research* 28 (1980) 1275–1289.
22. Eagle, J. N., Yee, J. R., An Optimal Branch, An optimal branch bound procedure for the constrained path, moving target search problem, *Operations Research* 38 (1990) 110–114.
23. Dell, R.F., Eagle, J.N., Martins, G.H.A., Santos, A.G., Using multiple searchers in constrained-path, moving target search problem, *Naval Research Logistics* 43 (1996) 463–480.
24. Hohzaki, R., Iida, K., Optimal strategy of route and look for the path constrained search problem with reward criterion, *European Journal of Operational Research* 100 (1997) 236–249.

BATCHING AND SCHEDULING WITH TARDINESS PENALTIES

Oncu Hazir · Safia Kedad-Sidhoum ·
Philippe Chrétienne

Abstract In this paper, we address batching and scheduling a single production system minimizing tardiness penalties. We define the optimal batching problem and scheduling problem, which is an integrated lot-splitting and scheduling problem. We examine the structural properties of the optimal solution and present some theoretical results. Using these results, an efficient dynamic programming algorithm is proposed.

1 Introduction

Production planning and control decisions are traditionally categorized into three levels: strategic, tactical and operational. In this research, we focus on the tactical and operational levels, and concentrate on lot-sizing, lot-streaming and scheduling problems. Lot-sizing, which is a tactical decision, identifies when and how much to produce so that the total cost is minimized. On the other hand, lot-streaming deals with splitting the production into batches with respect to the quantities of product planned. It is usually used for planning demands in multi-stage production systems so that the successive operations could overlap. By this way, lead times and work in process (WIP) inventories -control of which is crucial in just in time (JIT) production- could be reduced. There exists a large number of studies on lot-streaming and we refer the readers to the paper of [2] for a comprehensive survey. In addition to streaming, integration of a pegging and a scheduling phase that assigns lots to demands and sets the starting time of the lots respectively minimizing time or cost based criterion is valuable to generate globally optimal production plans.

Integrated lot-sizing, streaming and scheduling problems has been a research topic with practical implications, [3],[4]. In the scheduling literature, batch scheduling models, especially with set up times and costs, have been investigated by various researchers (see [1] for a comprehensive review). In these models, either job processing times are usually assumed to be fixed; or in some cases they are assumed to be controllable. When lot-streaming is incorporated, total processing time is fixed but early completion of the

Oncu Hazir, Safia Kedad Sidhoum, Philippe Chrétienne
Laboratoire d'Informatique de Paris 6, 104 Avenue du President Kennedy, 75016 Paris, France.
E-mail: oncu.hazir@lip6.fr, safia.kedad-sidhoum@lip6.fr, Philippe.Chretienne@lip6.fr

lots is possible. Potts and Van Wassenhove [5] make a review of integrated models that combine batching and lot-sizing decisions with scheduling. On the other hand, Selvarajah and Steiner [7] address the batching scheduling problem in a supply chain from the supplier's point of view. The aim is find the number of batches, batch sizes and the optimal schedule, while minimizing the total inventory holding and delivery cost.

Recently, Robert [6] combined lot-streaming and pegging, which addresses the transfer of materials between stocks and users, and used linear programming software to solve the derived models. In addition, she proposed a dynamic programming model that minimizes the sum of production costs and tardiness costs.

The problem could be defined as follows: We consider a single production system which can handle at most one customer order at a time and for which the nominal production rate is the same for all orders. Demand is deterministic and all orders are ready to be processed at time zero. We have K customer orders to satisfy. Each customer order k demands r_k units to be delivered at the due date d_k . If the order k is delivered after d_k , a penalty β_k , is paid. Two pegging conditions are considered: in *mono-pegging*, it is required that each customer order is met from a single batch, on the contrary in *multi-pegging*, orders could be satisfied from different batches. *Batch availability* is assumed, meaning that a product could not be released before its batch is completely processed. There exists lower and upper bound constraints, b and B , on the size of batch i . The starting and completion time of batch i will be denoted by S_i and C_i , respectively.

The cost of producing q_i^k units of order k in batch i is $\beta_k q_i^k T_i^k$ where $T_i^k = (C_i - d_k)^+$ is the tardiness of batch i with respect to due date d_k . The objective function is a quadratic function and it penalizes tardy deliveries. Then the problem is to find a production schedule with a minimum cost.

A production schedule Ψ could be defined as, (N, q, S) where N represents the total number of batches, and q defines the batch production variables with elements, q_i^k for $i = 1, \dots, N$ and $k = 1, \dots, K$, and S defines batch starting times. A feasible schedule must

(1) satisfy the batch size constraints:

$$b \leq \sum_{k=1}^K q_i^k \leq B \text{ for } i = 1, \dots, N, \text{ and}$$

(2) ensure that the time intervals $[S_i, S_i + \sum_{k=1}^K q_i^k]$ do not overlap, and

(3) ensure that all orders are met:

$$\sum_{i=1}^N q_i^k = r_k \text{ for } k = 1, \dots, K.$$

The objective is to find a feasible schedule that minimizes the total cost:

$$f(\Psi^*) = \text{Min} \left\{ \sum_{i=1}^N \sum_{k=1}^K \beta_k q_i^k (C_i - d_k)^+ \right\}$$

In the sequel, we address a practically relevant special case denoted with (P), multi-pegging with demand independent weights, i.e. $\beta_k = \beta$ for all $k = 1, \dots, K$. This case represents the case where customers are equivalently important. Problem (P), which we study in this research, encourages early deliveries and addresses scheduling the production systems in which partial delivery of orders are possible. Partial deliveries are widely encountered in industry because it prevents customer stock outs and improves customer service. Our model is applicable to model bottleneck stations and the results presented in this paper would serve as a basis to model more complicated production systems.

The organization of the paper is as follows: We first present some structural properties of an optimal schedule, then in order to solve (P), we propose a dynamic pro-

gramming based model that uses these properties. Finally, we discuss some possible extensions and future issues.

2 Structural Properties of an Optimal Schedule

Recently, Robert [6] investigated this problem (P) and defined the following properties:

Property 1. Processing the batches in a left justified single block without idle times is dominant.

Property 2. In multi-pegging with unitary weights, earliest due date (EDD) schedules are dominant.

Property 3. In multi-pegging, the batches with sizes in between b and $2b - 1$ are dominant when only tardiness cost is minimized.

In addition to the properties defined above, we present some novel structural properties of the optimal solution. Let us first introduce the following definitions:

A batch i is called *tardy* if it only consists of tardy units, i.e. $C_i > d_k$ for all $k = 1, \dots, K$ such that $q_i^k > 0$, and called *early* if it only consists of early units, i.e. $C_i < d_k$ such that $q_i^k > 0$ and *on time* if $C_i = d_k$ such that $q_i^k > 0$. In addition, we call a batch that processes only a single order *uniform*. We assume that the nominal unit production time is one unit time for any item so that the production quantity, q_i^k , also refers to the time required to process the customer order k in batch i . From now on, using property 2, without loss of generality (w.l.o.g.), we will number the customer orders according to the EDD rule and schedule accordingly, i.e. $k < l$ implies that $d_k \leq d_l$. Now we present some novel structural properties of an optimal schedule. Lemmas 1-4 address the batch sizing of the tardy batches.

2.1 Optimal Sizing of Tardy Batches

Lemma 1. There exists an optimal schedule where the consecutive tardy batches are ordered in non-decreasing batch size (SPT order).

Proof: First we will consider rescheduling two tardy consecutive batches. Let the optimal schedule Ψ have two consecutive tardy batches, namely batches i and j with $i < j$. Due to property 1, these two batches should be adjacent. Let us assume the following contradictory case, $q_i > q_j$:

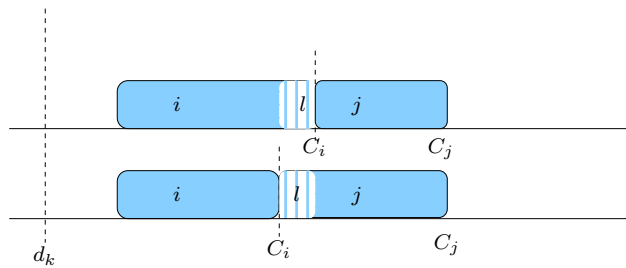


Fig. 1 Illustration of Lemma 1

A new schedule Ψ' could be generated by transferring the last unit of batch i (denote this unit by $[l]$) to batch j . As $b \leq q_j < q_i \leq B$ in schedule Ψ , feasibility will be maintained in schedule Ψ' , i.e. either $b < q'_j = q_j + 1 \leq q'_i = q_i - 1 < B$ or $b \leq q'_i = q_j \leq q'_j = q_i \leq B$. We illustrate both of the schedules in Figure 1. Each rounded rectangle represents a batch in the figure.

As a result of this operation, unit $[l]$ will complete later by an amount of q_j whereas all the remaining units in batch i will complete earlier by an amount of 1 time unit. The cost difference could be formulated as:

$$f(\Psi) - f(\Psi') = \beta (q_i - 1 - q_j) .$$

As $f(\Psi) - f(\Psi') \geq 0$, the transfer of the units until either $q_i \leq q_j$ or any order k in batch i becomes on time would not increase the costs.

On the other hand, for consecutive batches including more than two batches, the above mentioned transferring procedure could be reiterated. As a result, by a series of transfer operations among the adjacent batches, any schedule Ψ that includes non-SPT consecutive tardy batches could be converted to a schedule that satisfies the lemma (SPT order) without increasing the cost Ψ . □

Lemma 2. There exists an optimal schedule where the absolute difference between the sizes of any two tardy and uniform batches which process the same order is at most 1.

Proof: Let the optimal schedule Ψ have tardy uniform batches which process the customer order k . We know that this order is processed consecutively without any preemption (no idle time and no other order in between). As all these tardy batches would be consecutive, conditions of Lemma 1 applies as well. Among these batches, let us consider any two, namely batches i and j with $i < j$. Owing to Lemma 1, we can assume that $q_i \leq q_j$ w.l.o.g. In addition let us assume the following contradictory case for schedule Ψ , $q_i + 1 < q_j$:

We can generate a new schedule Ψ' by transferring one unit of batch j (denote this unit with $[l]$) to the batch i . Consequently, all the units in batches prior to batch j will complete later by an amount of 1 time unit, whereas unit $[l]$, will be tardy in batch i as well, because the units of the same order are tardy in batch i , but it completes earlier by an amount of $\sum_{k=i+1}^{k=j} q_k - 1$ time unit. We illustrate both of the schedules in Figure 2.

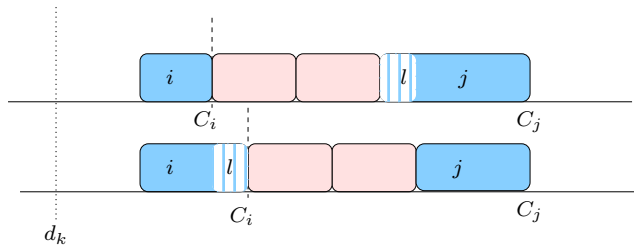


Fig. 2 Illustration of Lemma 2

As $b \leq q_i < q_i + 1 < q_j \leq B$ in schedule Ψ , feasibility will be maintained in schedule Ψ' , i.e. $b < q'_i = q_i + 1 \leq q'_j = q_j - 1 < B$. The cost difference would be:

$$f(\Psi) - f(\Psi') = \beta \left(\sum_{k=i+1}^{k=j} q_k - 1 - \sum_{k=i}^{k=j-1} q_k \right) = \beta (q_j - 1 - q_i) > 0.$$

Since $f(\Psi) - f(\Psi') > 0$, the transfer of the units would be profitable until either $q_j - 1 \leq q_i$ or the order k in batch i becomes on time. This contradicts the optimality of schedule Ψ . $\square$

Corollary 1: There exists an optimal schedule where the tardy batches that process the same order have the following structure: $q_j - 1 \leq q_i \leq q_j$ for any $i < j$.

Proof: Due to property 1, all the tardy batches which process the same order are consecutive. Using Lemma 1 and 2, the result follows. $\square$

Using Corollary 1, we propose the following lemma to construct the optimal schedule for tardy orders. In the sequel, we will denote the minimum and maximum number of batches to process Q units by $h_{min} = \lceil \frac{Q}{B} \rceil$ and $h_{max} = \lfloor \frac{Q}{b} \rfloor$ respectively. Furthermore, in the following lemmas, we assume that $\lceil \frac{Q}{B} \rceil \leq \lfloor \frac{Q}{b} \rfloor$, otherwise, the problem is infeasible [6].

Lemma 3. Given any time window $T = [t, t']$, if only Q units from a single customer order k are to be processed with h batches such that $h_{min} \leq h \leq h_{max}$ and $d_k \leq t$, there exists an optimal schedule in which, the first $h_1 = \lceil \frac{Q}{h} \rceil * h - Q$ batches have size $m_1 = \lceil \frac{Q}{h} \rceil - 1$ and the remaining $h_2 = h - h_1$ batches have size $m_2 = \lceil \frac{Q}{h} \rceil$.

Proof: As $d_k \leq t$ and a single order is processed, all the batches in T would be tardy and uniform and there exists no idle time between the batches. Since the conditions of Lemmas 1 and 2 is satisfied, there exists an optimal schedule Ψ that has the structure given in Corollary 1.

Considering Corollary 1, let us schedule the $Q' = Q - h_2$ units in the interval $T = [t, t']$ first. In this case, all the h batches should have equal size $m_1 = \lceil \frac{Q}{h} \rceil - 1$ and $b \leq m_1 < B$. The remaining h_2 extra units should be processed within the last h_2 batches, one unit in each batch. Otherwise, the conditions of Lemmas 1 and 2 would be violated. $\square$

We illustrate the structure with the following simple example:

Example 1. Given the following data $d_k = 10, Q = 17, T = [12, 30], b = 3, B = 5$ and $h = 4$, Lemma 3 results in the structure $h_1 = 3, h_2 = 1, m_1 = 4, m_2 = 5$ or 4, 4, 4, 5. Formally, starting from time 12, the first 3 batches have size of 4 whereas the last batch will have a size of 5.

Lemma 4. Given any time window $T = [t, t']$, if only Q units from a single customer order k is to be processed within $T = [t, t']$ and $d_k \leq t$, there exists an optimal schedule with $h_{max} = \lfloor \frac{Q}{b} \rfloor$ batches in T .

Proof:

Let the optimal schedule Ψ have $N_b < h_{max}$ batches and satisfies the conditions of the lemma. As all these batches would be tardy and a single customer order k is to be processed, conditions of Lemmas 1 and 2 apply. Let us integrate $h_{max} - N_b$ dummy batches with size 0 at the end of schedule Ψ . Starting from the last batch, the transfer of the units as explained in the proofs of Lemmas 1 and 2, and reiterating until conditions of Corollary 1 are satisfied results in a schedule with h_{max} batches without increasing the cost. Feasibility will be satisfied at the end, because according to Lemma 3, there will be $h_1 = \lceil \frac{Q}{h_{max}} \rceil * h_{max} - Q$ batches with size $m_1 = \lceil \frac{Q}{h_{max}} \rceil - 1$ and $h_2 = h - h_1$

batches of size $m_2 = \lceil \frac{Q}{h_{max}} \rceil$ at the end. Using the definition of h_{max} , if $h_1 > 0$, $b < \frac{Q}{h_{max}}$ and $b \leq m_1 < m_2 \leq B$. Therefore, schedule Ψ could be converted to a schedule with h_{max} batches without increasing the cost. $\square$.

Example 2. Consider the previous example, but without fixing the number of batches. In this case, according to Lemma 4, there exists an optimal schedule with $h_{max} = 5$ batches and the optimal structure would be 3, 3, 3, 4, 4.

2.2 Equivalence Properties of the Tardiness Cost

We will refer to the following Lemma in finding out the cost of tardy batches:

Lemma 5. The problem (P_1) to find the optimal schedule to process Q units of any customer order in time window $[t, t + Q]$ such that $t < d$ and the first batch completes after due date d is equivalent to the problem (P_2) to optimally schedule Q units in time window $[d, d + Q]$ so that the first batch contains at least $w = d - t$ units.

Proof: We denote cost functions of the problems P_1 and P_2 with f_1, f_2 and define $w = d - t$, and $\vartheta = \text{Max}\{b, w\}$. Let $T(Q)$ defines the optimal cost of processing Q units of a product in time window with length Q whose left hand is the deadline, i.e window $[d, d + Q]$, of the product.

On the other hand, if Q units start processing at time $d + u$, then the optimal cost could be reformulated as $\beta Qu + T(Q)$. Using this property and denoting the size of the first batch with x , then for each problem, we have:

$$\begin{aligned} f_1(Q) &= \text{Min}_{\vartheta \leq x \leq B} \{ \beta [x(x - w) + (Q - x)(x - w)] + T(Q - x) \} \\ &= \text{Min}_{\vartheta \leq x \leq B} \{ \beta Q(x - w) + T(Q - x) \}, \text{ where } x \text{ is the size of the first batch;} \\ f_2(Q) &= \text{Min}_{\vartheta \leq x \leq B} \{ \beta [x^2 + x(Q - x)] + T(Q - x) \} \\ &= \text{Min}_{\vartheta \leq x \leq B} \{ \beta Qx + T(Q - x) \}. \end{aligned}$$

As Q and w are given, P_1 and P_2 are equivalent. $\square$.

Corollary 2: The problem to find the optimal schedule to process Q units product in time window $[t, t + Q]$ where $d - b \leq t \leq d$ is equivalent to the problem to optimally schedule Q units in time window $[d, d + Q]$.

Proof: As $w = d - t < b$, then the first batch always completes after d and the result follows from Lemma 5. $\square$.

Note that we could use Lemma 3 to formulate $T(Q)$ as:

$T(Q) = \beta [\sum_{i=1}^{h_1} im_1^2 + \sum_{i=h_1+1}^h (h_1 m_1 + (i - h_1)m_2) * m_2]$ if $\lceil \frac{Q}{B} \rceil \leq \lfloor \frac{Q}{b} \rfloor$. Otherwise there is no feasible solution and a “big” cost value is assigned.

In what follows, to solve the problem P, we propose a dynamic programming model that uses all the properties defined above.

3 A Dynamic Programming Based Solution Algorithm

We describe a forward dynamic programming model has K stages that correspond to the customer orders. In an optimal production plan, some of the orders could share the same batch and because of the EDD rule, these batches could be either the first or the last batch of the order. The intermediate batches process units only from a single order hence they are uniform. We decompose each stage k and model the problem as an optimization problem to determine the size of a single shared batch, which contains units from order $k + 1$, and the sizes of intermediate uniform batches that process

units of order k . Let the variable x_{k+1} denote the number of units order $k + 1$ and forthcoming orders processed within the same batch with units of order k . For any order $k = 1, \dots, K$, if $b \leq r_k + x_{k+1}$, then units of order k could be processed in multiple batches. In this case, the decision variables x_k and y_k represent the number of units of order k to be processed within the same batch with units from the orders $k - 1$ and $k + 1$, respectively. This case and the corresponding decision variables are illustrated in Figure 3. From now on, we will denote cumulative orders shortly with R , i.e. $R_k = \sum_{i=1}^k r_i$.

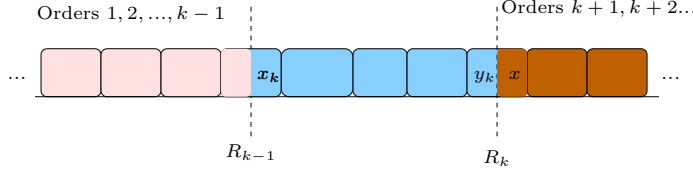


Fig. 3 Illustration of the Decision Variables

In the figure each rounded rectangle and color represent a batch and a customer order, respectively. For feasibility, we require $b \leq y_k + x_{k+1} \leq B$ for $k = 1, \dots, K$. An upper bound would be $UB = \min \{2b - 1, B\}$, hence $b \leq y_k + x_{k+1} \leq UB$. If the size of the batch is larger than $2b$, the batch could be split without increasing the costs of the schedule.

On the other hand, when $r_k + x_{k+1} < b$, lot-splitting is not feasible for order k and the shared batch includes units from order $k - 1$ as well. In this case, the units of order k are all early, all on time or all tardy. Let $h_k(x_{k+1})$ denote the minimum total cost of processing the first k orders given that the final batch of order k processes x_{k+1} units of order $k + 1$ and forthcoming orders. Therefore, the optimal cost will be referred as $h_n(0)$. We propose the following recursive model:

$$h_k(x_{k+1}) = \begin{cases} \min_{x_k, y_k} \{g_k(x_k, y_k, x_{k+1}) + h_{k-1}(x_k)\} & b \leq r_k + x_{k+1} \\ \beta r_k(R_k + x_{k+1} - d_k) + h_{k-1}(x_{k+1} + r_k) & r_k + x_{k+1} < b, d_k < R_k + x_{k+1} \\ h_{k-1}(x_{k+1} + r_k) & r_k + x_{k+1} < b, d_k \geq R_k + x_{k+1} \end{cases}$$

Note that $g_k(\cdot)$ is defined for the case $b \leq r_k + x_{k+1}$ and refers to the total cost of units of order k . We will consider the following four cases to define $g_k(\cdot)$. In these cases, $y_k \leq UB$ and $x_k \leq UB$ for all $k = 1, \dots, K$. In addition, we remind the assumption that the nominal unit production time is one unit time for any item and $T_k(Q)$ is the optimal cost of processing Q units of customer order k by starting at the due date d_k .

Case 1: If $d_k < R_{k-1} + x_k$ then, all the units of order k would be tardy. This case is illustrated in Figure 4.

Using the equivalence property of Corollary 2,
 $g_k(x_k, y_k, x_{k+1}) = \beta[x_k(R_{k-1} + x_k - d_k) + (r_k - x_k - y_k)(R_{k-1} + x_k - d_k)] + T_k(r_k - x_k - y_k) + \beta y_k(R_k + x_{k+1} - d_k)$.
 $= \beta(R_{k-1} + x_k - d_k)(r_k - y_k) + T_k(r_k - x_k - y_k) + \beta y_k(R_k + x_{k+1} - d_k)$.

Case 2: If $R_{k-1} + x_k \leq d_k \leq R_k - y_k$, then some of the intermediate batches could be early, some on time and some could be tardy. To differentiate these, we need the decision variable, w_k . The variable w_k denotes the units of the order k processed

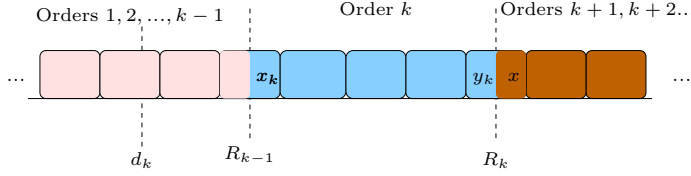


Fig. 4 Illustration of Case 1

in first batch completed after d_k and which have been completed before or at the due date d_k . The cost function would be:

$$g_k(x_k, y_k, x_{k+1}) = \text{Min}_{0 \leq w_k \leq UB} \{ \nu_k(x_k, y_k, x_{k+1}, w_k) \} + \beta y_k (R_k + x_{k+1} - d_k).$$

We consider two sub-cases to define the cost function $\nu_k(\cdot)$:

Case 2.1: $0 \leq w_k < b$

This case is illustrated in Figure 5.

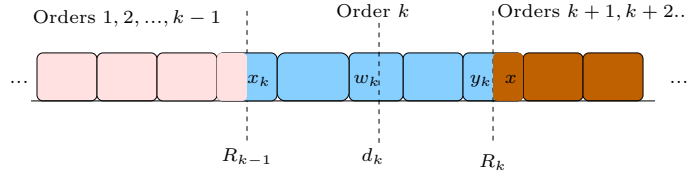


Fig. 5 Illustration of Case 2

Using the equivalence property of Corollary 2,

$$\nu_k(x_k, y_k, x_{k+1}, w_k) = E_k(d_k - w_k - R_{k-1} - x_k) + T_k(R_k - y_k - d_k + w_k) - \beta w_k (R_k - y_k - d_k + w_k).$$

In the above equation, $E_k(Q)$ denotes the optimal cost of processing Q units of customer order k before its due date d_k .

$E_k(Q) = 0$ if $\lceil \frac{Q}{b} \rceil \leq \lfloor \frac{Q}{b} \rfloor$, otherwise the problem becomes infeasible and we assign a “big” cost value.

Case 2.2: $b \leq w_k < UB$

In this case, we need the variable z_k to denote the units in the same batch but completed after the due date d_k . W.l.o.g. we can assume that $b \leq w_k + z_k \leq UB$, for all $k = 1, \dots, K$. This case is illustrated in Figure 6.

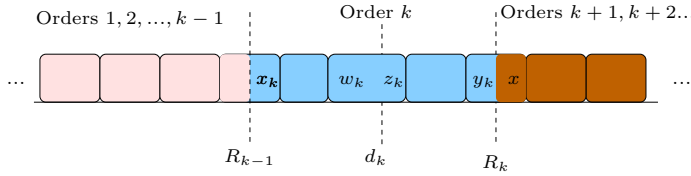


Fig. 6 Illustration of Case 2.2

The variable z_k is required to be set as follows:

$$\nu_k(x_k, y_k, x_{k+1}, w_k) = \text{Min}_{0 \leq z_k \leq UB - w_k} \{ E_k(d_k - w_k - R_{k-1} - x_k) + \beta [z_k(z_k + w_k) + z_k(R_k - d_k - z_k - y_k)] + T_k(R_k - d_k - z_k - y_k) \}.$$

$$= \text{Min}_{0 \leq z_k \leq UB - w_k} \{E_k(d_k - w_k - R_{k-1} - x_k) + \beta[z_k(w_k + R_k - d_k - y_k)] + T_k(R_k - d_k - z_k - y_k)\}.$$

Case 3: If $R_k - y_k < d_k < R_k + x_{k+1}$, then all the intermediate batches would be early. This case is illustrated in Figure 7. The cost would be:
 $g_k(x_k, y_k, x_{k+1}) = E_k(r_k - x_k - y_k) + \beta y_k(R_k + x_{k+1} - d_k).$

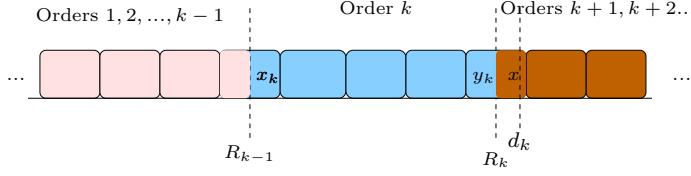


Fig. 7 Illustration of Case 3

Case 4: If $R_k + x_{k+1} \leq d_k$ then, all the units of order k would be early and
 $g_k(x_k, y_k, x_{k+1}) = E_k(r_k - x_k - y_k).$

This case is illustrated in Figure 8.

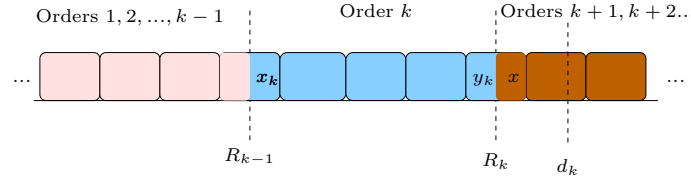


Fig. 8 Illustration of Case 4

Initial Conditions: $x_1 = x_{K+1} = y_K = 0$; $h_0(\cdot) = 0$.

Algorithmic Complexity: For the problem, we consider K stages, corresponding to the customer orders. Each stage could be solved in at most UB^4 steps so that the complexity for each stage is $O(b^4)$. Considering all the stages and the complexity of sorting to satisfy EDD order, our DP algorithm provides an optimal solution in $O(K(\log K + b^4))$ time.

To illustrate the problem and the solution algorithm, we solve the simple example with the following parameters and present the optimal solution.

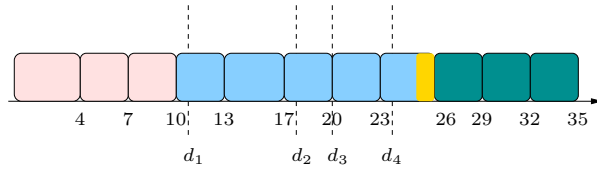


Fig. 9 An Illustrative Example

An Illustrative Example: We consider scheduling four customer orders with multi-pegging given the following data: $K = 4; b = 3; B = 5; r_1 = 10, r_2 = 15, r_3 = 1, r_4 = 9; d_1 = 11, d_2 = 18, d_3 = 20, d_4 = 24$.

The optimal schedule is illustrated in Figure 9 below. Each rounded rectangle represents a customer order following the EDD rule and completion time of each batch is illustrated in the time line below.

4 Conclusions

In this paper, we have investigated a production planning problem with tardiness penalties for a single production system. In order to solve the problem, we have followed an integrated lot-streaming and scheduling approach. We have examined the structural properties of the optimal solution, presented theoretical results, and developed a dynamic programming algorithm that extensively makes use of these properties.

As a future extension of this research, we will incorporate the setup times and costs and earliness penalties. Considering the real world requirements, integration of the setup constraints is important. Furthermore, modeling both the earliness and tardiness penalties addresses the JIT requirement of lower lead times and WIP, therefore is practically relevant especially for just in time systems.

References

1. A. Allahverdi, C.T. Ng, T.C.E. Cheng, and M.Y. Kovalyov, A survey of scheduling problems with setup times or costs, *European Journal of Operational Research* 187(3), 985-1032,(2008).
2. Chang J.H. and Chiu H.N., A comprehensive review of lot-streaming, *International Journal of Production Research* 43 (8), 1515-1536 (2005).
3. Dauzère-Pérès S. and Lasserre J.B. On the importance of sequencing decisions in production planning and scheduling. *International Transactions in Operational Research*, 9(6), 779-793,(2002).
4. Drexel A., and A Kimms, Lot sizing and scheduling: Survey and extensions, *European Journal of Operational Research* 99, 221-235 (1997).
5. Potts C.N. and L.N. Van Wassenhove, Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity. *Journal of the Operational Research Society* 43 395-406 (1992).
6. Robert A. Optimization des batches de production. PhD thesis, Université Pierre et Marie Curie (Paris 6)(2007).
7. Selvarajah, E. and Steiner, G. Batch scheduling in a two-level supply chain: A focus on the supplier. *European Journal of Operational Research* 173 226-240 (2006).

Bicriteria models to minimize the total weighted number of tardy jobs with convex controllable processing times and common due date assignment

Dvir Shabtay · George Steiner

Abstract We study a single-machine scheduling problem in a flexible framework where both job processing times and due dates are decision variables to be determined by the scheduler. The model can also be applied for quoting delivery times when some parts of the jobs may be outsourced. We analyze the problem with common due date assignment and a convex resource consumption function. We provide a bicriteria analysis where the first criterion is to minimize the total weighted number of tardy jobs plus due date assignment cost, and the second criterion is to minimize total weighted resource consumption. We consider four different models for treating the two criteria. Although the problem of minimizing a single integrated objective function can be solved in polynomial time, we prove that the three bicriteria models are $\mathcal{NP}$ -hard. We also present special cases which are polynomially solvable and frequently occur in practice.

1 Introduction

Manufacturing or service organizations must frequently quote delivery dates (due dates) to clients. Customers demand that suppliers meet contracted delivery dates or face substantial penalties. For example, Slotnick and Sobel [14] cite contracts from the aerospace industry, which may impose tardiness penalties of millions of dollars on subcontractors for aircraft components. In order to avoid tardiness penalties, including the possibility of losing customers, suppliers are under increasing pressure to quote *attainable* delivery dates for customer orders. Naturally, longer due dates are easier to meet, but promising delivery dates too far into the future may not be acceptable to the customer. At the same time, shorter due dates increase the probability that the order will be delivered late. Thus there is an important tradeoff between assigning relatively short due dates to customer orders and avoiding tardiness penalties, which creates the

Dvir Shabtay
Department of Industrial engineering and Management, Ben-Gurion University of the Negev,
Beer-Sheva, Israel
E-mail: dvirs@bgu.ac.il

George Steiner
DeGroote School of Business, McMaster University, Hamilton, Ontario, Canada L8S 4M4
E-mail: steiner@mcmaster.ca

need for methodology that allows firms to quote attainable delivery dates and obtain efficient schedules at the same time.

While traditional scheduling models considered due dates as given by exogenous decisions (see Baker and Scudder [1]), in a more agile system where due date assignment and scheduling are integrated, they are determined by taking into account the system's ability to meet the quoted delivery dates. For this reason, an increasing number of recent studies viewed due date assignment as part of the scheduling process and showed how the ability to control due dates can be a major factor in improving system performance. The most frequently studied model is the *common* due-date assignment method (denoted by *CON*), for which all jobs are assigned the same due date, that is $d_j = d$ for $j = 1, \dots, n$, where d_j is the due date of job j and $d \geq 0$ is a decision variable [5]. Similarly to due dates, job processing times in scheduling were traditionally considered to be fixed parameters. However, a supplier may have several alternative ways to supply and deliver products to the customer. For example, the supplier may outsource part of the production to subcontractors if this allows them to quote an earlier delivery date. This type of outsourcing can be modeled by assuming that the jobs have controllable processing times and the job processing times can be shortened by employing additional resources via outsourcing. There are also many other situations where the scheduler can shorten the processing time of a job by allocating more resources to it, see e.g., Janiak [6] and [7], which describes an interesting application of a scheduling model with controllable processing times in steel mills. Due to the large variety of applications, there is extensive literature on the subject of scheduling with controllable processing times. A survey of results up to 1990 is provided by Nowicki and Zdrzalka [11], and more recent ones are given by Chudzik *et al.* [2], Shabtay and Steiner [13] and Janiak *et al.* [8].

In this paper, we study scheduling problems in a flexible environment where both the job processing times and due date assignment are controllable by the scheduler. The decisions to be made incur two types of costs: the scheduling costs, which include the costs of due date assignment and penalties for tardy jobs, and the cost of controlling the job processing times. In [12] we studied a model minimizing the *sum* of all these costs. However, there are many situations where these two types of costs are not comparable or additive. For example, we may have a preset budget for the resource consumption costs or the maximum amount of resource that is available, or the maximum amount that can be subcontracted, and thus these costs have to be controlled separately from the scheduling costs. This results in having to schedule with two separate criteria.

1.1 Problem Definition

Our bicriteria scheduling problem is formally stated as follows: a set of n independent, non-preemptive jobs, $J = \{1, \dots, n\}$, are available for processing at time zero and are to be processed by a system modeled as a single machine. The processing time of job j , p_j , is a continuous convex function of the amount of resource that is allocated to the processing of the job given by the following *resource consumption function*:

$$p_j(u_j) = (\theta_j/u_j)^k, \quad j = 1, \dots, n, \quad (1)$$

where n is the number of non-preemptive jobs, u_j is a decision variable that represents the *amount* of resource allocated to the processing of job j , θ_j is a positive parameter,

which represents the *workload* of the processing operation for job j and k is a positive constant. We note that many scheduling models use linear resource consumption functions, however, these models fail to reflect the *law of diminishing marginal returns*. This law states that productivity increases at a decreasing rate with the amount of resource employed. In order to model this, many studies on scheduling with resource allocation assume that the job processing time is a convex decreasing function of the amount of resource allocated to the job, described by eq. (1). This model has been used extensively in continuous resource allocation theory [13]. The due dates are assignable by the CON method. A schedule S is defined by the sequence in which the jobs are processed, a resource allocation vector $\mathbf{u}^* = (u_1, u_2, \dots, u_n)$, and a common due date d . The quality of a schedule S is measured by two criteria: The scheduling criterion, which includes (weighted) penalties for tardy jobs and the cost of due date assignment, defined by

$$Z(S) = \sum_{j=1}^n \alpha_j U_j + \beta \sum_{j=1}^n d_j; \quad (2)$$

and the second criterion is the total resource consumption cost (or amount) given by

$$V(S) = \sum_{j=1}^n v_j u_j, \quad (3)$$

where for schedule S , $C_j(S)$ is the completion time of job j and U_j is the tardiness indicator variable for job j , i.e., $U_j = 1$ if $C_j(S) > d_j$ and $U_j = 0$ if $C_j(S) \leq d_j$. In addition, α_j is the cost of job j being tardy, β is the cost of one unit of delivery time quotation per job and v_j is the cost of one unit of resource allocated to job j . Note that for the CON method, eq. (2) can be converted to

$$Z(S) = \sum_{j=1}^n \alpha_j U_j + \beta n d. \quad (4)$$

Since our problem has two criteria, four different optimization problems can arise:

- The first one, which we denote by P1, is to minimize $F_l(Z, V) = \sum_{j=1}^n \alpha_j U_j + \beta \sum_{j=1}^n d_j + \sum_{j=1}^n v_j u_j$. Using the scheduling notation introduced in [15], this problem can also be referred to as $1|d_j = d \text{ unknown, conv}|F_l(Z, V)$;
- The second one, denoted by P2, is to minimize $\sum_{j=1}^n \alpha_j U_j + \beta \sum_{j=1}^n d_j$ subject to $\sum_{j=1}^n v_j u_j \leq V$; where V is a limitation on the total resource consumption cost. Following the notation in [15], we refer to this problem as $1|d_j = d \text{ unknown, conv}|\epsilon(Z/V)$;
- The third one, which we denote by P3, is to minimize $\sum_{j=1}^n v_j u_j$ subject to $\sum_{j=1}^n \alpha_j U_j + \beta \sum_{j=1}^n d_j \leq K$, where K is a given upper bound on the scheduling cost. We refer to this problem by $1|d_j = d \text{ unknown, conv}|\epsilon(V/Z)$ (based on [15]);
- The last one, denoted by P4 (and referred to by $1|d_j = d \text{ unknown, conv}|\#(V/Z)$), is to identify the set of Pareto-optimal schedules (points) (V, Z) , where a schedule S with $V = V(S)$ and $Z = Z(S)$ is called *Pareto-optimal* (or *efficient*) if there does not exist another schedule S' such that $V(S') \leq V(S)$ and $Z(S') \leq Z(S)$ with at least one of these inequalities being strict.

It should be noted that solving P4 also solves P1-P3 as a by-product. Note also that all problem types involve both continuous decisions (the due date and processing time values) and combinatorial (sequencing) decisions. Similar bicriteria models were analyzed in [10] with linear resource consumption functions.

A linear time optimization algorithm was presented in [12] for the P1-type scheduling problem $1|d_j = d \text{ unknown}|conv|F_l(Z, V)$. The complexity and practical solvability of the P2-P4 bicriteria problems, however, remained an open question until now. The main goal of this paper is to provide answers for these open questions.

1.2 Organization of the Paper

The rest of the paper is organized as follows. In Section 2 we determine the optimal continuous resource allocations as functions of the job partition into early and tardy jobs and thus reduce the P2 problem to a pure combinatorial problem. In contrast with the polynomial solvability of the P1 problems, we prove in Section 3 that all P2-P4 problems are $\mathcal{NP}$ -hard. In Section 4 we give a polynomial time algorithm for special cases of these problems that are important in practice.

2 Reducing the P2 Problem to a Partition Problem

With fixed processing times, our P2 problem becomes a $1|d_j = d \text{ unknown}|\sum_{j=1}^n \alpha_j U_j + \beta n d$ due date assignment problem. This problem was analyzed by De *et al.* [3] and Kahlbacher and Cheng [9], who reduced it to a partition problem where set J is to be partitioned into two disjoint subsets:

- (a) Subset E , that will be sequenced at the beginning of the schedule, containing the early jobs whose total processing time is used to set the common due date value.
- (b) Subset T , consisting of the tardy jobs, which will be sequenced after the completion of subset E .

They presented the following $O(n)$ time optimization algorithm for the solution of this partition problem.

Algorithm 1: Optimization algorithm for the CON due date assignment problem with fixed job processing times.

Step 1. Define $E = \{j \in J \mid p_j \leq \alpha_j / (\beta n)\}$ as the set of early jobs and $T = J \setminus E$ as the set of tardy jobs.

Step 2. Schedule set E before set T , where the sequence within each set is immaterial.

Step 3. The optimal due date coincides with the completion time of set E , i.e., $d = \sum_{j \in E} p_j$.

We can see from Algorithm 1 that the optimal due date assignment and job partition into $E \cup T$ are determined by the processing times. For our problem with controllable processing times, this means that they depend on the resource allocation strategy. Furthermore, the value of the scheduling objective in eq. (4), under an optimal due date assignment strategy, can be represented as follows:

$$Z(\tau, \mathbf{u}) = \sum_{j \in T} \alpha_j + \beta n \sum_{j \in E} p_j(u_j) = \sum_{j \in T} \alpha_j + \beta n \sum_{j \in E} (\theta_j / u_j)^k, \quad (5)$$

where τ is the partition of the jobs into the two sets E and T (also dependent on $\mathbf{u}$). Thus, our P2 problem is reduced to finding τ and $\mathbf{u}$ which minimize the objective in eq. (5) subject to $\sum_{j=1}^n v_j u_j \leq V$.

In the following Lemma, we present the optimal resource allocation strategy for the P2 problem as a function of an assumed job partition into early and tardy subsets.

Lemma 1 *For an arbitrary partition $\tau = E \cup T$, the optimal resource allocation $\mathbf{u}^*(\tau) = (u_1^*(\tau), u_2^*(\tau), \dots, u_n^*(\tau))$ for the $1|d_j = d \text{ unknown, conv}| \epsilon(Z/V)$ problem is*

$$u_j^*(\tau) = \begin{cases} \frac{\theta_j^{k/(k+1)} \times V}{v_j^{1/(k+1)} \sum_{i \in E} (v_i \theta_i)^{k/(k+1)}} & \text{if } j \in E \\ 0 & \text{if } j \in T \end{cases} \quad (6)$$

Proof For an arbitrary partition τ , our problem reduces to a continuous resource allocation problem of minimizing $\beta n \sum_{j \in E} p_j(u_j) = \beta n \sum_{j \in E} (\theta_j / u_j)^k$ subject to $\sum_{j=1}^n v_j u_j \leq V$. Since the objective value is independent of u_j for $j \in T$, it implies that $u_j^* = 0$ for $j \in T$. In addition, since each $p_j(u_j)$ is a decreasing continuous function of u_j , the $\sum_{j=1}^n v_j u_j \leq V$ constraint will be satisfied as an equality. Thus, the optimal resource allocation for each job in set E can be obtained by applying the Lagrangian method (described in detail in the Appendix). Since both the objective and the constraint are convex functions, the resulting resource allocation in eq. (6) is optimal.

By substituting (6) into the objective in (5), we obtain that the objective value under an optimal resource allocation strategy and as a function of τ can be written as

$$Z(\tau, \mathbf{u}^*(\tau)) = \sum_{j \in T} \alpha_j + \frac{\beta n}{V^k} \left(\sum_{j \in E} (\theta_j v_j)^{k/(k+1)} \right)^{k+1}, \quad (7)$$

reducing our problem to the purely combinatorial problem of finding the partition τ which minimizes (7).

3 The $\mathcal{NP}$ -hardness of Problems P2-P4

Let us first define the following decision problem associated with our optimization problems (P2-P4):

DVP1: Given a single-machine scheduling problem where job due dates are assignable according to the CON method, job processing times are controlled by the convex resource consumption function (1), and parameters K and V . Is there a schedule with $\sum_{j=1}^n \alpha_j U_j + \beta n d \leq K$ and $\sum_{j=1}^n v_j u_j \leq V$?

It is easy to see that **DVP1** is in $\mathcal{NP}$, since if we have a job partition τ and a resource allocation vector $\mathbf{u} = (u_1, u_2, \dots, u_n)$, we can easily check in polynomial time whether both $\sum_{j=1}^n \alpha_j U_j + \beta n d \leq K$ and $\sum_{j=1}^n v_j u_j \leq V$. We next show that **DVP1** is $\mathcal{NP}$ -complete by reducing the $\mathcal{NP}$ -complete **Partition** problem to it.

Partition: Given a finite set $\mathbf{A} = \{w_1, w_2, \dots, w_h\}$ of positive integers, where $\sum_{j=1}^h w_j = W$. Can the set A be partitioned into two disjoint subsets, A_1 and A_2 , so that $\sum_{j \in A_i} w_j = W/2$ for $i = 1, 2$?

Theorem 1 *DVP1 is $\mathcal{NP}$ -complete even if $v_j = 1$ for $j = 1, \dots, n$.*

Proof We construct the following instance of **DVP1** from an instance of **Partition**: There are $n = h$ jobs, where the job processing times follow eq. (1) with $\theta_j = w_j^2$ for $j = 1, \dots, n$. The penalty for completing job j after its due date is $\alpha_j = w_j$ and the resource consumption cost is identical for all jobs with $v_j = 1$ for $j = 1, \dots, n$. In addition, $k = 1$, β is an arbitrary positive integer and the limitations are $K = 3W/4$ and $V = \beta nW$.

We will first show that if we have a YES instance of **Partition**, then there exists a schedule for the corresponding instance of **DVP1** with $\sum_{j=1}^n \alpha_j U_j + \beta n d \leq K = 3W/4$ and $\sum_{j=1}^n u_j \leq V = \beta nW$. Let E be the set of jobs corresponding to set A_1 of the solution of **Partition**, and let T be the set of all other jobs.

Substituting eq. (6), the optimal resource allocation strategy with the given $V = \beta nW$ is

$$u_j^*(\tau) = \begin{cases} 2\beta n w_j & \text{if } j \in E \\ 0 & \text{if } j \in T. \end{cases} \quad (8)$$

It is clear that $\sum_{j \in J} u_j^*(\tau) = \sum_{j \in E} u_j^*(\tau) = \sum_{j \in E} 2\beta n w_j = \beta nW$. In addition, by eq. (7), we have

$$Z(\tau, \mathbf{u}^*(\tau)) = \sum_{j \in T} w_j + \frac{1}{W} \left(\sum_{j \in E} w_j \right)^2 = \frac{W}{2} + \frac{W}{4} = \frac{3W}{4}. \quad (9)$$

Conversely, we show that if we have a NO instance of **Partition**, then there is no schedule for the corresponding instance of **DVP1** with $\sum_{j=1}^n \alpha_j U_j + \beta n d \leq K = 3W/4$ and $\sum_{j=1}^n u_j \leq V = \beta nW$. Let us assume that, contrary to the statement we want to prove, there is a schedule for the corresponding instance of **DVP1** with $\sum_{j=1}^n \alpha_j U_j + \beta n d \leq 3W/4$ and $\sum_{j=1}^n u_j \leq V = \beta nW$, and let E and T denote the set of early and tardy jobs, respectively, in this schedule. Since we have a NO instance of **Partition**, then either $\sum_{j \in E} w_j > W/2$ or $\sum_{j \in E} w_j < W/2$.

If $\sum_{j \in E} w_j > W/2$, then let $\Delta = \sum_{j \in E} w_j - W/2 > 0$. For this case, we have $Z(\tau, \mathbf{u}^*(\tau)) = \frac{W}{2} - \Delta + \frac{1}{W} (W/2 + \Delta)^2 = \frac{W}{2} - \Delta + \frac{1}{W} (W^2/4 + W\Delta + \Delta^2) = \frac{3W}{4} + \frac{\Delta^2}{W}$. This value is greater than $3W/4$ since $\Delta > 0$, which contradicts our assumption. If $\sum_{j \in E} w_j < W/2$, then let $\Delta = W/2 - \sum_{j \in E} w_j > 0$. For this case, we have $Z(\tau, \mathbf{u}^*(\tau)) = \frac{W}{2} + \Delta + \frac{1}{W} (W/2 - \Delta)^2 = \frac{W}{2} + \Delta + \frac{1}{W} (W^2/4 - W\Delta + \Delta^2) = \frac{3W}{4} + \frac{\Delta^2}{W}$, which is again greater than $3W/4$, and thus contradicts again our assumption and completes the proof.

Note that **DVP1** is the decision version of both the P2-type $1|d_j = d \text{ unknown}, \text{conv}| \epsilon(Z/V)$ and the P3-type $1|d_j = d \text{ unknown}, \text{conv}| \epsilon(Z/V)$ problem. In addition, since the P4-type $1|d_j = d \text{ unknown}, \text{conv}| \#(V/Z)$ problem is at least as hard as the P2-type and P3-type problems, we have the following corollary.

Corollary 1 *The P2-type $1|d_j = d \text{ unknown}, \text{conv}| \epsilon(Z/V)$, P3-type $1|d_j = d \text{ unknown}, \text{conv}| \epsilon(V/Z)$ and the P4-type $1|d_j = d \text{ unknown}, \text{conv}| \#(V/Z)$ problems are all $\mathcal{NP}$ -hard even if all $v_j = 1$.*

4 Polynomially solvable cases

Lemma 2 Consider jobs l and m satisfying the following conditions: $v_l\theta_l \leq v_m\theta_m$ and $\alpha_l \geq \alpha_m$. Then for any V value, there is an efficient schedule in which job l is scheduled before m .

Proof Consider an efficient schedule S , let jobs l and m satisfy the conditions of the lemma and assume that m is before l in S . Interchange l and m to get schedule $\tilde{S}$. If both l and m belong to E or to T in S , then changing their order does not affect the optimality of the schedule since the scheduling objective value in (5) and the optimal resource allocation given by (6) are independent of the internal sequence within each set. If l and m belong to different sets, then by Step 2 of Algorithm 1, we can assume that $m \in E$ and $l \in T$ in schedule S , and by Lemma 1 that $u_l(S) = 0$. Define the resource allocations in $\tilde{S}$ according to eq. (6). Since the optimal resource allocations defined by (6) satisfy (13) of the Appendix, we also have $\sum_{j=1}^n v_j u_j(\tilde{S}) = \sum_{j \in E(\tilde{S})} v_j u_j(\tilde{S}) = V$, where $E(\tilde{S}) = E \setminus \{m\} \cup \{l\}$. Let $E' = E(\tilde{S}) \setminus m$. Then based on (7), the change in the scheduling objective function is

$$Z(S) - Z(\tilde{S}) = \alpha_l - \alpha_m + \frac{\beta n}{V^k} [(B + (v_m\theta_m)^{k/k+1})^{k+1} - (B + (v_l\theta_l)^{k/k+1})^{k+1}],$$

where $B = \sum_{j \in E'} (v_j\theta_j)^{k/k+1}$. Since $v_l\theta_l \leq v_m\theta_m$ and $\alpha_l \geq \alpha_m$, we have that $Z(S) - Z(\tilde{S}) \geq 0$. This means that $\tilde{S}$ is also an efficient schedule.

Lemma 2 implies that we can restrict our search for efficient schedules in which every pair of jobs satisfying the lemma's conditions also satisfies the *precedence constraint* $l \preceq m$, i.e., l is scheduled before m . It also follows that in any algorithm that tries to find the set E of an efficient schedule, we can consider job l before job m for inclusion in the set E .

In the following we present a polynomial time optimization algorithm to determine the set of *all* efficient schedules for the special case where the partial order $\preceq$ becomes a linear (complete) order, i.e., when the two orders appearing in Lemma 2 are agreeable and order *all* the jobs in the same sequence. These conditions will be satisfied in many applications, since they simply say that for shorter jobs (jobs with lower workload and resource consumption cost), the supplier has to pay larger tardiness penalties if they are late, and this coincides with the typical customer expectation that it should be easier to meet a due date for shorter jobs than for longer ones. For the agreeable case, Lemma 2 implies that the early jobs must form an initial subsequence of jobs ordered into a chain by $\preceq$. The following algorithm finds the efficient frontier for the agreeable case, starting with the largest possible early set $E = J$.

Algorithm 2: Optimization algorithm for the $1|d_j = d \text{ unknown, conv, agreeable}| \#(V/Z)$ problem.

Step 0. Preprocessing: Sort and renumber the jobs according to $\preceq$ generated by the agreeable condition.

Step 1. Initialization: Set $q = n$ where q denotes the number of early jobs in the current segment of efficient schedules, $E = \{1, \dots, n\}$, $T = \phi$. Let $V_0^{\lim} = 0$ and $V_{n+1}^{\lim} = \infty$, where $V_j^{\lim}$, $j = 1, \dots, n+1$, denotes the minimum total weighted resource consumption needed for an efficient schedule that has exactly the first j jobs scheduled early in the $\preceq$ order.

Step 2. Calculate the minimum amount of total weighted resource consumption for job q (but not job $q + 1$) to be in set E by

$$V_q^{\text{lim}} = \left(\frac{\beta n}{\alpha_q} \times \left(\left(\sum_{j=1}^q (v_j \theta_j)^{k/(k+1)} \right)^{k+1} - \left(\sum_{j=1}^{q-1} (v_j \theta_j)^{k/(k+1)} \right)^{k+1} \right) \right)^{1/k}. \quad (10)$$

Step 3. The q -th segment of the Pareto curve is given by

$$\left(V, \sum_{j=q+1}^n \alpha_j + \frac{\beta n}{V^k} \left(\sum_{j=1}^q (\theta_j v_j)^{k/k+1} \right)^{k+1} \right) \quad (11)$$

for any V satisfying $V_q^{\text{lim}} < V \leq V_{q+1}^{\text{lim}}$. The optimal resource allocation for any point in this segment is given by eq. (6), and the optimal common due date is computed by

$$d^* = \sum_{j \in E} p_j(u_j^*) = \sum_{j=1}^q p_j(u_j^*) = \left(\sum_{j=1}^q (\theta_j v_j)^{k/k+1} \right)^{k+1} V^{-k},$$

where the last equality can be derived by substituting (6) into (1).

Step 4. Set $q = q - 1$. If $q = 0$, then stop. Otherwise go to step 2.

Algorithm 2 starts by renumbering the jobs according to the optimal job sequence that corresponds to the agreeable condition. In step 2 the algorithm determines the minimum amount of total weighted resource consumption for job q so that in the corresponding efficient schedule $E = \{1, 2, \dots, q\}$. The algorithm starts with the largest initial segment in $\preceq$, i.e., with $E = J$. Using eq. (7), V_q^{lim} is computed as the value of the total weighted resource consumption at which we are indifferent between assigning job q to be the next tardy job or leaving it in the early set, i.e., V_q^{lim} is the value at which these two scenarios have equal scheduling costs. Thus, V_q^{lim} satisfies

$$\sum_{j=q}^n \alpha_j + \frac{\beta n}{(V_q^{\text{lim}})^k} \left(\sum_{j=1}^{q-1} (\theta_j v_j)^{k/k+1} \right)^{k+1} = \sum_{j=q+1}^n \alpha_j + \frac{\beta n}{(V_q^{\text{lim}})^k} \left(\sum_{j=1}^q (\theta_j v_j)^{k/k+1} \right)^{k+1},$$

whose solution yields the V_q^{lim} in eq. (10). Thus, for any $V_q^{\text{lim}} < V \leq V_{q+1}^{\text{lim}}$ the early set includes jobs $1, \dots, q$ while the tardy set consists of jobs $q+1, \dots, n$. Therefore, according to eq. (7), the efficient curve segment for $V_q^{\text{lim}} < V \leq V_{q+1}^{\text{lim}}$ is given by eq. (11).

Theorem 2 *Algorithm 2 solves the $1|d_j = d \text{ unknown, conv, agreeable}|\#(V/Z)$ problem in $O(n \log n)$ time.*

Proof The correctness of Algorithm 2 follows from Lemmas 1, 2 and the preceding discussion. Step 0 requires sorting and thus takes $O(n \log n)$ time; Steps 2 and 3 can be performed in linear time in the first iteration. In all other iterations, however, these steps require only constant time. Since Step 4 also takes constant time, and there are n iterations of Steps 2-4, the overall complexity of Algorithm 2 is $O(n \log n)$ indeed.

Corollary 2 *The efficient frontier (Pareto curve) for the $1|d_j = d \text{ unknown, conv, agreeable}|\#(V/Z)$ problem contains at most n curve segments.*

Proof Each iteration through Step 2 and Step 3 constructs one curve segment of the tradeoff curve, and there are at most n such iterations before the algorithm ends.

5 Conclusions and Future Research

We considered four different versions of the problem of minimizing the weighted number of tardy jobs plus due date assignment cost and minimizing the total weighted resource consumption in scheduling a single machine. We assume that each job processing time is a bounded, decreasing convex function of the number of resource units that are allocated to the job and all jobs share the same common due date, which is also a decision variable. In an earlier study, Shabtay and Steiner [12] showed that the problem of minimizing an integrated objective (formed as the sum of the two criteria) is solvable in polynomial time. In contrast to this result, we prove that all of the other three versions of the problem are $\mathcal{NP}$ -hard. It is very interesting to notice that the polynomial solvability of P1 and the $\mathcal{NP}$ -hardness of P4 lead to the conclusion that in fact the enumeration of the supported strict Pareto optima is polynomial and the enumeration of non-supported strict Pareto optima is hard (see [15]) for more details). This is quite an unusual result in multicriteria scheduling. In future research, we may seek approximation and exact algorithms to solve problems P2-P4.

Acknowledgements This research was partially supported by the Paul Ivanier Center for Robotics and Production Management, Ben-Gurion University of the Negev and by the Natural Sciences and Engineering Research Council of Canada under grant No. OG1798-03.

References

1. Baker, K.R. and Scudder, G.D., Sequencing with earliness and tardiness penalties: a review, *Operations Research*, **38**, 22-36 (1990).
2. Chudzik, K., Janiak, A., and Lichtenstein, M., Scheduling problems with resource allocation. In A. Janiak, ed., *Scheduling in computer and manufacturing systems*, WKL, Warszawa (2006).
3. De, P., Ghosh, J.B., and Wells, C.E., Optimal delivery time quotation and order sequencing, *Decision Sciences*, **22**, 379-390 (1991).
4. Graham, R.L., Lawler, E.L., Lenstra, J.K. and Rinnooy Kan, A.H.G., Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, **5**, 287-326 (1979).
5. Gordon, V., Proth, J.M., and Chu, C.B., A survey of the state-of-the-art of common due date assignment and scheduling research, *European Journal of Operational Research*, **139**, 1-25 (2002).
6. Janiak, A., One-machine scheduling with allocation of continuously-divisible resource and with no precedence constraints, *Kybernetika*, **23**(4), 289-293 (1987).
7. Janiak, A., Minimization of the makespan in a two-machine problem under given resource constraints, *European Journal of Operational Research*, **107**, 325-337 (1998).
8. Janiak, A., Janiak, W., and Lichtenstein, M., Resource management in machine scheduling problems: a survey, *Decision Making in Manufacturing and Services*, **1**, 59-89 (2007).
9. Kahlbacher, H.G., and Cheng, T.C.E., Parallel machine scheduling to minimize costs for earliness and number of tardy jobs, *Discrete Applied Mathematics*, **47**, 139-164 (1993).
10. Leyvand, Y., Shabtay, D., and Steiner, G., Optimal delivery time quotation to minimize total tardiness penalties with controllable processing times, submitted (2008).
11. Nowicki, E., and Zdrzalka, S., A survey of results for sequencing problems with controllable processing times, *Discrete Applied Mathematics*, **26**, 271-287 (1990).
12. Shabtay, D., and Steiner, G., Optimal due date assignment and resource allocation to minimize the weighted number of tardy jobs on a single machine, *Manufacturing & Service Operations Management*, **9**(3), 332-350 (2007).
13. Shabtay, D., and Steiner, G., A survey of scheduling with controllable processing times, *Discrete Applied Mathematics*, **155**(13), 1643-1666 (2007).

14. Slotnick, S.A. and Sobel, M.J., Manufacturing lead-time rules: customer retention versus tardiness costs, *European Journal of Operational Research*, **169**, 825-856 (2005).
15. T'kindt, V., and Billaut, J.-C., *Multicriteria scheduling: theory, models and algorithms*. 2nd edition, Springer, Berlin (2006).

Appendix - Optimal Resource Allocation Strategy as a Function of τ

Since under an optimal resource allocation strategy, the total weighted resource consumption constraint is satisfied as equality, we can apply the Lagrangian method to solve the resource allocation problem. The Lagrangian function is

$$L(u_1, u_2, \dots, u_n, \mu) = \beta n \sum_{j \in E} (\theta_j / u_j)^k + \mu \left(\sum_{j \in E} v_j u_j - V \right), \quad (12)$$

where μ is the Lagrangian multiplier. Differentiating eq. (12) with respect to the decision variables (u_j for $j = 1, \dots, n$ and μ), the necessary and sufficient conditions for an optimal solution are:

$$\frac{\partial L(u_1, u_2, \dots, u_n, \mu)}{\partial \mu} = \sum_{j \in E} v_j u_j - V = 0; \quad (13)$$

$$\frac{\partial L(u_1, u_2, \dots, u_n, \mu)}{\partial u_j} = -\frac{k\beta n \theta_j^k}{u_j^{k+1}} + \mu v_j = 0 \quad \text{for } j \in E. \quad (14)$$

Using eq. (14) we obtain that

$$\frac{\theta_i^k}{v_i u_i^{k+1}} = \frac{\theta_j^k}{v_j u_j^{k+1}} = \frac{\mu}{k\beta n} \quad \text{for } i, j \in E. \quad (15)$$

Thus,

$$u_j = \left(\frac{\theta_j}{\theta_i} \right)^{\frac{k}{k+1}} \left(\frac{v_i}{v_j} \right)^{\frac{1}{k+1}} u_i \quad \text{for } i, j \in E. \quad (16)$$

Inserting eq. (16) into eq. (13) we obtain that

$$\sum_{j \in E} v_j \left(\frac{\theta_j}{\theta_i} \right)^{\frac{k}{k+1}} \left(\frac{v_i}{v_j} \right)^{\frac{1}{k+1}} u_i = V, \quad (17)$$

or equivalently that

$$u_i \theta_i^{\frac{-k}{k+1}} v_i^{\frac{1}{k+1}} \sum_{j \in E} (\theta_j v_j)^{\frac{k}{k+1}} = V. \quad (18)$$

Rearranging eq. (18) we obtain the solution in eq. (6).

Heuristic approach to the airline schedule disturbances problem: multi-fleet case

Slavica Dožić, Milica Kalić, Obrad Babić, Mirjana Čangalović

Abstract In order to reduce negative effects of flight schedule disturbance, such as meteorological conditions, aircraft failure, etc. a decision support system (DSS) – multi-fleet case (ASDP-MFC) is developed aiming to assist the decision makers in handling disturbances in a real time. This system is based on a heuristic algorithm which generates a list of different feasible schedules ordered according to the value of an objective function. The possibilities of DSS are illustrated by real numerical examples that concern middle size European airline's flight schedule disturbances.

1 Introduction

The daily schedule is the final version of the flight schedule for the considered day, which takes into account all the changes that were undertaken during the period from its publication till just before its realization. As the flight schedule is realised, disturbances occur that cannot be predicted by the airline in advance. When disturbances make it impossible to realize the planned flight schedule, the dispatcher at the airline operational centre (AOC) defines a new daily operational flight schedule. The airline schedule disturbance problem – multi-fleet case (ASDP-MFC), which is faced by dispatchers at the AOC day-to-day, is considered in this paper. This research presents extended work on the Airline Schedule Optimization project [2] in which the basic ASDP model is developed. During the ASDP-MFC solving the dispatcher at the AOC considers airline policy and defines a new flight schedule that will optimize an objective function and satisfy the corresponding constraints. Depending on airline policy, when designing the new flight schedule, the dispatcher delays or cancels some flights and reassigns flights to available aircraft in order to reduce the negative effects of disturbance.

The airline flight schedule designing in the case of disturbances is a problem that has been addressed in literature with several different models, objective functions, constraints and assumptions, as well as different solution approaches. Stojković [11] and Teodorović and Stojković [13] developed several models of airline schedule recovery problems and the corresponding exact methods and heuristics for solving them. Stojković et al. [12] developed a model which can be solved to optimality in a real-time, but only in cases of small disturbances. Rakshit et al. [9] presented a decision support system incorporating the models considered by

First Author: Slavica Dožić

Affiliation: University of Belgrade - The Faculty of Transport and Traffic Engineering

E-mail: s.dozic@sf.bg.ac.rs

Second Author: Milica Kalić

Affiliation: University of Belgrade - The Faculty of Transport and Traffic Engineering

E-mail: m.kalic@sf.bg.ac.rs

Third Author: Obrad Babić

Affiliation: University of Belgrade - The Faculty of Transport and Traffic Engineering

E-mail: o.babic@sf.bg.ac.rs

Fourth Author: Mirjana Čangalović

Affiliation: University of Belgrade - The Faculty of Organizational Sciences

E-mail: cangalovic.mirjana@fon.rs

Jarrah et al. [6]. Rosenberger et al. [10] presented an optimization model that reschedules legs and reroutes aircraft by minimizing rerouting and cancellation costs. Andersson and Värbrand [3] and Andersson [4] suggested using heuristics and meta-heuristics for solving the Flight Perturbation Problem. Abdelghany et al. [1] presented an integrated decision-support tool for airlines schedule recovery during irregular operations, which includes a schedule simulation model and a resource assignment optimization model. Kalić and Pavković [7] developed a special heuristic algorithm for solving the schedule disturbances problem and model based on two objective functions. Nedeljković [8] presented a developed model and heuristic algorithm which offers several solutions as a response to disturbances.

This paper presents a mathematical formalization for the airline schedule disturbances problem and a developed heuristic algorithm for generating new daily operational flight schedules. The corresponding DSS is developed aiming to assist the dispatcher in handling disturbances in a real time. The mathematical model is defined in such a way that by changing the objective function, it can support different decision-making strategies of the airline. The proposed heuristic algorithm offers a list of different feasible solutions ordered according to the value of an objective function. During the decision-making process the dispatcher could define a new strategy, resolve a problem and select a solution. The paper has five sections: the introduction and a literature review are followed by the definition of the problem and its mathematical formalization in Section 2. A heuristic algorithm proposed for its solution is described in Section 3. In the fourth and fifth sections a numerical example and conclusions, including further research recommendations are presented.

2 Problem definition and mathematical formalization

In the case when a disturbance has occurred at an airport or on an aircraft and consequently some flights can no longer be realized according to the planned daily schedule, a new departure time shall be defined and/or a new aircraft shall be assigned to each such a flight, or it shall be cancelled, in such a way that the objective function is minimized. This function represents the additional costs of the airline caused by disturbances. In this paper the additional costs consist of six elements: the aircraft swapping cost, the priority and the non-priority flight cancellation cost, the passengers delay cost, the aircraft regular maintenance disturbance cost, and the aircraft balance cost. In order to overcome disturbances and find a new daily operational flight schedule, some assumptions are introduced:

- A disturbance may occur at an airport or on an aircraft. All disturbed airports and aircraft are identified and the corresponding starting time for a new flight schedule is defined.
- The flight schedule recovery period, for which a new flight schedule is to be designed, is the time period from the starting time, corresponding to the identified disturbance, to the last moment at which the timetable for the next day is not disturbed.
- A rotation is a sequence of flights where the first flight in the series departs from the base airport, and the last one arrives at the base airport.
- The airline has a fleet which consists of different types of aircraft characterized by different seat capacities, where aircraft of the same type have the same seat capacity; all aircraft can be divided in three categories: aircraft for short-haul (cat. 1), aircraft for mid-haul (cat. 2) and aircraft for long-haul flights (cat. 3). All three categories consist of different aircraft types; the aircraft ground handling time depends on the aircraft type and the airport where the handling occurs.
- Aircraft can be swapped – bigger aircraft can service flights originally assigned to smaller ones (seat capacity is not smaller) and if is from the same or the first following category in relation to originally assigned aircraft.
- There are no spare aircraft in the fleet; a set of priority flights is given (flights with the slot time, transfer passengers, etc.); the maximal allowed delay is defined for each flight; ferry flights (flights without passengers) are not allowed; crew constraints are not considered.

- Priority flights can be realized according to the so-called VIA principle. If flights i and j are planned flight, where i is a priority flight from airport B to airport C, while j is a non-priority one from airport A to airport C and if at airport B the aircraft assigned to flight i has a failure, then the aircraft assigned to flight j can realize an additional unplanned flight i' (from A to B), and then priority flight i . In this way that aircraft can service both flight i and flight j , but only if its seat capacity is not smaller than the total number of passengers on these two flights. Therefore, for each planned priority flight (flight i) a set of additional non planned flights (flights i'), which can be used for its realization applying the VIA principle, is given in advance. Obviously, such an additional flight should have the destination airport identical to the origin airport of the priority flight. Also, there should exist at least one non-priority flight (flight j) with the origin and the destination airports identical to that of the origin airport of the additional flight and the destination airport of the priority flight, respectively. The cost of passenger's delay of priority flight (if delay exists) calculates as product of priority flight delay, sum of passengers on both flights and average passenger's delay cost per time unit.
- For each flight the average passenger's delay cost per time unit (equal for all flights and all passengers) and the cost of aircraft type swapping are known in advance. The cancellation costs are different for priority and non-priority flights.

Starting with the previous assumptions, the following constraints of the problem ASDP-MFC are specified: (1) *Time constraints*: Each flight in a new flight schedule should not depart earlier than the known departure time planned before the disturbance occurrence. If the departure time of a flight in a new schedule is delayed with respect to the planned time, this delay should not be greater than the maximal delay allowed for this flight. The time period between two consecutive flights in the rotation of an aircraft should be sufficient for its ground handling. The take-off and landing of each flight in the new schedule should be serviced at the corresponding airport by the end of its working hours, i.e. by its closing time valid on the day for which this schedule is designed. (2) *Aircraft maintenance constraints*: Each aircraft, planned to have the regular technical maintenance at the end of the flight schedule recovery period, should finish its rotation at the defined airport where the maintenance is performed. (3) *Aircraft balance constraints*: In order to successfully service the planned flights after the recovery period, it is necessary that at the end of this period certain types of aircraft are available at each airport in sufficient numbers (Otherwise, the disturbance will be extended to the following day). This condition can be satisfied by requiring that, for each of such types, a given number of aircraft with the seat capacity not smaller than the seat capacity characterizing this type, should finish their rotation at the given airport. (4) *Capacity constraints*: The number of passengers on each flight should not be greater than the seat capacity of the aircraft assigned to this flight. The seat capacity of an aircraft, used to service a priority flight applying the VIA principle, should not be smaller than the total number of passengers on that flight and a non-priority flight that is indirectly realized in this way.

In order to make the problem definition more precise and easier to understand, we will give a mathematical formalization of the objective function and the main constraints of the problem ASDP-MFC.

Let us introduce the following notations of its input parameters: F : a set of flights which should be serviced in the recovery period; Pr : set of priority flights, $Pr \subset F$; $VIA(i)$, $i \in Pr$: set of additional flights which can be used in the realization of priority flight i using the VIA principle, $VIA(i) \cap F = \emptyset$; $id(i, i')$, $i \in Pr$, $i' \in VIA(i)$: a non-priority flight which can be indirectly realized by servicing flights i' and i using the VIA principle, $id(i, i') \in F$; F' : set F enlarged with all additional flights which can be used for the VIA realization of priority flights, i.e. $F' = F \cup \bigcup_{i \in Pr} VIA(i)$; AP : set of airports; $o(i)$, $d(i)$, $i \in F'$: the origin and the destination airport

of flight i , $o(i)$, $d(i) \in AP$; $TP(i)$, $i \in F$: the departure time of flight i planned before the disturbance occurrence; $delay(i)$, $i \in F$: the maximal allowed delay of flight i ; $pax(i)$, $i \in F$: the number of passengers on flight i ; $k(i)$, $i \in Pr$: the cancellation cost of priority flight i ; $k_2(i)$, $i \in F \setminus$

Pr : the cancellation cost for non-priority flight i ; k_l : the delay cost of a flight per a time unit; k_p : the cost for passenger's delay per time unit; $d(i, l)$: the cost of servicing flight i by aircraft of type l ; $l \in TYPE$; $j^*(i)$: the aircraft which should service flight i according to rotation plan; AC : set of aircraft available for servicing flights from F' ; $TYPE$: set of aircraft types; $atype(j)$, $j \in AC$: the type of aircraft j , $atype(j) \in TYPE$; $cap(l)$, $l \in TYPE$: the seat capacity of an aircraft of type l ; $cat(l)$, $l \in TYPE$: the category of an aircraft of type l ;

$$cat(l) = \begin{cases} 1, & \text{short - haul} \\ 2, & \text{mid - haul} \\ 3, & \text{long - haul} \end{cases}, \quad l \in TYPE$$

$a(l, k)$, $l \in TYPE$, $k \in AP$: the ground handling time for an aircraft of type l at airport k ; $dis(j)$, $j \in AC$: the airport where aircraft j is located at the starting time of the recovery period, $dis(j) \in AP$; $t(i, j)$, $i \in F'$, $j \in AC$: block time (the time between engine start at the airport of origin and engine stop at the airport of destination) or the duration of flight i serviced by aircraft j ; MNT : set of aircraft which should finish its rotation at certain airports for the regular technical maintenance, $MNT \subset AC$; $mnt(j)$, $j \in MNT$: the airport where aircraft j should finish its rotation, $mnt(j) \in AP$; $clo(k)$, $k \in AP$: the closing time of airport k valid on the day for which a new schedule is designed; $PTYPE(k)$, $k \in AP$: set of aircraft types which should be available at airport k at the end of the recovery period; $notype(l, k)$, $k \in AP$, $l \in PTYPE(k)$: the number of aircraft with the seat capacity not smaller than $cap(l)$, which should finish their rotation at airport k ; $pen(l, k)$, $k \in AP$, $l \in PTYPE(k)$: the penalty cost per aircraft when the number of aircraft, with the seat capacity not smaller than $cap(l)$, which finish their rotation at airport k , is smaller than $notype(l, k)$; DAP : set of airports with detected disturbances, $DAP \subset AP$; $TDAP(k)$, $k \in DAP$: the earliest possible time when airport k can start operation after the disturbance elimination; DAC : set of aircraft with detected disturbances, $DAC \subset AC$; $TDAC(j)$, $j \in DAC$: the earliest possible departure time of aircraft j from airport $dis(j)$ after the disturbance elimination;

The variables of the problem ASDP can be defined in the following formal manner.

- $rot(1, j), rot(2, j), \dots, rot(l(j), j)$: the rotation of aircraft j , where $l(j)$ is the number of flights in this rotation, $rot(l, j) \in F'$ for $l=1, 2, 3, \dots, l(j)$, and $j \in AC$;
- $TR(i)$, $i \in F'$: the earliest possible departure time for flight i ;
- $X(i, j)$, $i \in F'$, $j \in AC$: a binary variable equal to 1 if aircraft j is assigned to flight i , 0 otherwise; $can(i)$, $i \in F$: a binary variable equal to 1 if flight i is cancelled, 0 otherwise; $ment(i)$, $j \in MNT$: a binary variable equal to 1 if aircraft j , requiring the regular technical maintenance, has not finished its rotation at airport $mnt(j)$, 0 otherwise; $sat(s, l, j, k)$, $k \in AP$, $j \in AC$, $l \in PTYPE(k)$, $s \in \{1, 2, \dots, notype(l, k)\}$: a binary variable equal to 1 if aircraft j , finishing its rotation at airport k , satisfies the s -th necessity of this airport for aircraft type l , 0 otherwise.

Using the previously introduced notations the objective function of ASDP-MFC can be formally expressed by relation (1):

$$\begin{aligned} \min F = & \sum_{i \in F} \sum_{j \in AC} (d(i, atype(j)) - d(i, atype(j^*(i)))) X(i, j) + \sum_{i \in Pr} k(i) can(i) + \\ & + \sum_{i \in F \setminus Pr} k_2(i) can(i) + k_p \sum_{i \in F} (TR(i) - TP(i)) pax(i) (1 - can(i)) + \\ & + \sum_{j \in MNT} k_3(j) ment(j) + \sum_{k \in A} \sum_{l \in PTYPE(k)} pen(l, k) (notype(l, k) - \sum_{s=1}^{notype(l, k)} \sum_{j \in AC} sat(s, l, j, k)) \end{aligned} \quad (1)$$

The first term in (1) represents the cost of aircraft type swapping, while the other terms are the priority-flight cancellation cost, the non-priority flight cancellation cost, the passenger's delay cost, the aircraft maintenance disturbance cost and the balance disturbance cost, respectively. Let us notice that, defining the objective function in the form (1), we in fact relax the aircraft maintenance and the aircraft balance constraints, introducing the total

penalized violation of these constraints as a part of its cost. In this way, although the constraints can be unsatisfied, the maximization of the objective function tends to minimize their violation.

The main constraints of ASDP-MFC can be formally defined by (2)-(14).

$$TR(i) \geq TP(i), \quad \text{for } i \in F, \quad (2)$$

$$TR(i) - TP(i) \leq \text{delay}(i), \quad \text{for } i \in F, \quad (3)$$

$$TR(\text{rot}(l,j)) + t(\text{rot}(l,j),j) + a(\text{atype}(j),d(\text{rot}(l,j))) \leq TR(\text{rot}(l+1,j)), \text{ for } l=1, 2, \dots, l(j)-1, j \in AC, \quad (4)$$

$$d(\text{rot}(l,j)) = o(\text{rot}(l+1,j)), \quad \text{for } l=1, 2, \dots, l(j)-1, j \in AC, \quad (5)$$

$$o(\text{rot}(l,j)) = \text{dis}(j), \quad \text{for } j \in AC, \quad (6)$$

$$TR(i) \leq \text{clo}(o(i)), \quad \text{for } i \in F', \quad (7)$$

$$TR(i) + t(i,j) \leq \text{clo}(d(i)), \quad \text{for } i \in F', j \in AC, X(i,j)=1, \quad (8)$$

$$TR(i) \geq \text{TDAC}(j), \quad \text{for } j \in DAC, X(i,j)=1, \quad (9)$$

$$TR(i) \geq \text{TDAP}(o(i)), \quad \text{for } o(i) \in DAP, \quad (10)$$

$$TR(i) \geq \text{TDAP}(d(i)) - t(i,j), \text{ for } d(i) \in DAP, X(i,j)=1, \quad (11)$$

The conditions (2)-(11) express the time constraints of the problem: (2) and (3) mean that each flight departs not earlier than the planned departure time, while its delay is not greater than the maximal allowed value. Constraints (4) indicate that the following flight in the rotation of an aircraft cannot take off before the previous flight has landed and the aircraft has been ground-handled. Equalities (5) and (6) provide that in the rotation the destination airport of a flight and the origin airport of the following flight are identical, as well as the first flight starts from the airport where the aircraft is located stays at the starting time of the recovery period. Inequalities (7) and (8) express that a flight should take-off before the closing of its origin airport and it should not land after the closing of its destination airport. Also, a flight cannot take off before repairing a breakdown of the aircraft assigned to it (9). In the case when its origin airport has a detected disturbance, a flight cannot take off before the airport is reopened (10), while if its destination airport is disturbed, it cannot land before the disturbance has been eliminated (11).

$$\text{cap}(\text{atype}(j)) - \text{cap}(l) \geq 0, \quad \text{for } \text{sat}(s,l,j,k)=1, \quad k \in AP, \quad j \in AC, \quad l \in \text{PTYPE}(k), \quad s \in \{1, 2, \dots, \text{notype}(l,k)\} \quad (12)$$

Inequalities (12) are related to the aircraft balance constraints and provide that each aircraft, which satisfies the need for a certain aircraft type at an airport to have the seat capacity not smaller than the seat capacity characterizing this type.

$$\text{cap}(\text{atype}(j)) \geq \text{pax}(i), \quad \text{for } i \in F, j \in AC, X(i,j)=1, \quad (13)$$

$$\text{cap}(\text{atype}(j)) \geq \text{pax}(\text{id}(i,i')) + \text{pax}(i), \quad \text{for } i \in Pr, i' \in \text{VIA}(i), X(i,j)=1 \quad (14)$$

Conditions (13) and (14) are the capacity constraints: (13) means that the number of passengers on each flight is not greater than the seat capacity of the assigned aircraft, while (14) expresses that the total number of passengers on that flight and an indirectly realized non-priority flight is not greater than the seat capacity of the assigned aircraft, when a priority flight is realized using the VIA principle.

$$\text{cap}(\text{atype}(j)) \geq \text{cap}(\text{atype}(j^*(i))), \quad \text{for } i \in F, j \in AC, X(i,j)=1, \quad (15)$$

$$0 \leq \text{cat}(\text{atype}(j)) - \text{cat}(\text{atype}(j^*(i))) \leq 1, \quad \text{for } j \in AC, i \in F, X(i,j)=1 \quad (16)$$

The flights assigned to an aircraft, in the case of disturbance, can be performed only by aircraft which capacity is not smaller than the capacity of originally aircraft (15), and if that new aircraft is from the same or from the first following category (16).

The mathematical modelling of ASDP-MFC, defined by (1)-(16) is only partial, hence many constraints, which are assumed to be satisfied, are not formalized. For example, the objective function (1) is correct only if we assume that an aircraft cannot be assigned to a cancelled flight, i.e. $\text{can}(i)=1$, if and only if $X(i,j)=0$ for each $j \in AC$. But, this mathematical formalization could be a basis for developing more sophisticated mathematical models of the ASDP problem, such as a mixed integer programming or a constraint programming model. As this problem is known to be NP-hard [5] and it should be solved in a real time, instead of making a great effort to find such an appropriate model and try to solve it using an exact method, we focus on a heuristic approach to the problem. Therefore, in the next section we

propose a special heuristic technique for determining a list of feasible “satisfactory” (sub-optimal) new daily flight schedules, among which the dispatcher can select and implement the most convenient one.

3 Heuristic algorithm for ASDP-MFC

Before presenting the steps of a proposed heuristic algorithm for ASDP-MFC, we will describe in more details some basic notions introduced in Section 2. In this paper rotation is defined as a one-day rotation or part of multi-day rotation during a considered day. Rotations consist of mini rotations and simple rotation segments. A mini rotation is a series of flights attached to each other where the departure airport of the first flight is the same as the arrival airport of the last flight in the series (A-B-A). A simple segment of the rotation is the series of flights attached to each other where the departure airport of the first flight and the arrival airport of the last flight in the series are different airports (A-B-C).

The results of disturbances are flight delay and/or cancellation. The flight can be cancelled temporarily or permanently. A temporarily cancelled flight can be realized by adding its cancelled mini rotation/simple segment to the rotation of some of the other aircraft (considering departure airport, arrival airport and time constraints) before the first flight in the rotation, between flights in the rotation, or after the last flight in the rotation. The temporarily cancelled flights and its cancelled mini rotations/simple segments are permanently cancelled if there is no possibility to add them to the rotation of one of the other aircraft by any of previously described ways. Also, a priority flight and its mini rotation or all following flights of its simple segment by the end of the considered day, are permanently cancelled if there is no possible way to realize it using the VIA principle.

In order to reduce the total delay and/or the balance disturbance cost, the delayed flights can be crossed. Crossing delayed flights can be achieved by two operations: removing part of one and adding it to the other rotation, and interchanging parts of two rotations. Removing part of one and adding to the other rotation refer to the delayed flights whose delay is not directly caused by the airport disturbance. Shifting those flights to the other rotation is an attempt to reduce its delay, i.e. to decrease the value of the objective function. Interchanging parts of two rotations is possible only if it leads to a delay reduction. In order to interchange some parts of two rotations, it is necessary that these parts depart from the same airport within the time period from the planned departure time to the time caused by the maximal allowed delay of the considered flights. Let us mention that not only parts with the delayed flights can be interchanged, but also with delayed and no delayed flights if the total delay is reduced in this way. A special heuristic algorithm consists of the following steps.

Step 1 - Designing the basic feasible schedule: In order to create a new feasible daily operational flight schedule, all operating aircraft are considered, both aircraft that have landed at their arrival airports and aircraft that are in flight at the moment of the disturbance and which capacity is not smaller than the capacity of disturbed aircraft, on condition that the aircraft is from the same or from the first following category of aircraft. For each disturbed aircraft whose rotation does not contain priority flights, a new feasible schedule is designed as follows:

- If the delay is less than the maximal allowed, the basic feasible solution is designed by shifting the delayed flight by the delay time. The following flight departs either on time or after completing the previous flight.
- If the delay of flight is greater than allowed maximum, the basic feasible solution is designed so that the mini rotation containing the delayed flight is temporarily cancelled. If the other flights in that rotation do not have a delay greater than the allowed maximum, they are realized with or without delay. Mini rotations/simple segments of rotation with allowed delay are also temporarily cancelled if the constrain related to airport working hours is violated.

- If the flight belongs to the simple segment of rotation, and delay is greater than allowed maximum, all flights until the end of that rotation are temporarily cancelled.

For each disturbed aircraft whose rotation contains a priority flight, a new feasible schedule is designed as follows:

- If delay of the priority flight is not greater than the maximum allowed one, then this flight is realized with this delay, while a following flight in the rotation depart either on time or it is shifted to start immediately after the completion of the previous flight.
- If the delay of the priority flight is greater than the maximal allowed one and there is a mini rotation(s) which precedes this flight, then this mini rotation(s) is temporarily cancelled, until the priority flight is serviced with the allowed delay.
- If the priority flight is part of a mini rotation assign the mini rotation of the considered priority flight to other aircraft from the same or from the first following category of aircraft.
- If it is not possible to find an aircraft which can realize the priority flight's mini rotation, find an aircraft which can realize the priority flight using the VIA principle and which is from the same or from the first following category of aircraft as the originally assigned aircraft. When the priority flight is a part of simple segment of rotation, realize it in the same, abovementioned way.
- If none of the abovementioned procedure led to realization of the priority flight, cancel it permanently. If the considered aircraft has more than one priority flight in its rotation, repeat the previous procedure for each of them.

Now the complete basic feasible schedule is designed by repeating the entire Step 1 for each aircraft that is influenced by the disturbance, and the corresponding value of the objective function value is calculated. If there are no temporarily cancelled flights in the designed complete basic schedule, the heuristic algorithm goes to Step 3. Otherwise it goes to Step 2.

Step 2 - Adding temporarily cancelled flights: All temporarily cancelled flights are sorted in a list according to the following procedure: partition the set of all temporarily cancelled flights into the cancelled mini rotations and simple segments and sort them in the four groups. The groups are: (1) mini rotations that begin and end at the base airport of their rotations; (2) mini rotations that begin and end at some other airport; (3) simple segments that begin or end at the base airport of their rotation; (4) simple segments that neither begin nor end at the base airport of their rotations. After that lexicographically sort elements (mini rotations or single segments) of each group according to the decreasing total number of flights and the decreasing total number of passengers. Starting from the top of the list of the temporarily cancelled flights, the heuristic algorithm passes through all its elements and tries to add each of them to the basic schedule, determined in Step 1. The element (a mini rotation or a simple segment) should be added to the schedule within the time period between the planned departure time and the latest possible departure time (according to the maximal allowed delay or airport working hours) of the first flight in this element. The element is added to the rotation of an aircraft of the same or the first following category as the aircraft that was originally supposed to realize it in order to reduce total number of cancelled flights and additional costs. When the list of the temporarily cancelled flights is exhausted, the algorithm goes to Step 3 if there are the delayed flights in the current schedule. Otherwise, it stops.

Step 3 - Rotation crossing: All mini rotations and simple segments of the schedule, obtained by Step 2, which contain delayed flights, are sorted into the list of the delayed flights according to the total delay costs of passenger. The delay cost of passengers calculates as a product of number of passengers on flight, delay of flight obtained in the step 2 and average delay cost per minute for each delayed flight. Rotation crossing refers to removing parts of the rotation and adding them to some other rotation (the first part of Step 3) and interchanging parts of two rotations (the second part of Step 3) in order to reduce passenger's delay costs. In this step the aircraft balance and the number of realized flights are not changed. As a result of Step 3 the final n solutions list is offered to the dispatcher, sorted in a descending order of the objective function value. Then go to Step 4.

Step 4: The end of the algorithm.

4 Numerical example

As a result of the model application, at least one feasible solution is provided. Each solution is presented by a graphical representation of the daily schedule and table which includes planned and obtained data on the flight schedule (flight delay, flight cancellation, swapping registration or type of aircraft, etc.).

The algorithm is illustrated with reference to middle size European airline's timetable. The chosen values of penalties represent the airline policy and in numerical examples they are: penalty for priority flight cancellation $k(i)=200,000$ units per flight, penalty for non-priority flight cancellation $k_2(i)=100,000$ units per flight, penalty for aircraft balance disturbance $pen(l,k)=3,000$ units per aircraft, penalty for passenger delay per minute $k_p=1$ unit per passenger minute. In the considered example maximum allowed delay for domestic flight is 360 minutes and maximum allowed delay for international flight is 180 minutes. Within the given day, operations were executed by 29 aircraft (9 different aircraft types), assigned to 126 flights. Because there is no data about aircraft maintenance within the technical maintenance system for this day, it was assumed that all aircraft introduced into the realization of scheduled operations were available till the end of given day and the constraint related to the technical base is not considered. The costs of realization flight by appropriate aircraft type are calculated for each flight, and they are put, together with the other flight data, in the software data base. The solutions list offered to the dispatchers consists of a maximum of 10 solutions.

Example: Aircraft 11 is failed and it is being repaired from 14:00 till 20:00. This aircraft is planned to realize one priority flight 415 (A-C). Due to the limited space, graphical presentation of input and output data is omitted, while output results are given in table form. The software offered 6 solutions, but only first three solutions will be illustrated.

Solution 1: The disturbance directly affected priority flight's mini rotation A-C-A (priority flight 415 and non-priority flight 416). In the first step of the algorithm, this mini rotation is assigned to aircraft 12, where both flights are realized on time, without delay. Flight 307 (A-P) will be delayed 123 minutes, because of servicing abovementioned mini rotation. There are no temporarily cancelled flights in the first step of the algorithm, so we move to the third step. In the first part of step 3, delayed flights are added in the rotation of the aircraft from the same or the first following category in order to reduce the value of objective function. In this case, the flight delayed in the step 1, is added to aircraft 13 where it is realized on time. All flights are realized on time, so there are no passenger's delay costs, and value of objective function is equal to 0 (Table 1).

Table 1. Data on flights influenced by disturbance, solution 1

Flight	415	416	307
Departure airport	A	C	A
Arrival airport	C	A	P
Planned departure time	14:14	17:09	17:53
Real departure time	14:14	17:09	17:53
Delay (min)	0	0	0
A/C planned to realize the flight	11	11	12
A/C which actually realized the flight	12	12	13
Number of passengers on flight	137	136	145
Value of objective function	0		

Solution 2: The second solution is obtained after the first part of the step 3. In this solution also there are no delayed flights. In the first part of the step 3 (adding flights), flight 307 is assigned to aircraft 23 instead of 12. Because of aircraft type changing (flight 307) the cost of planned aircraft type swapping is appeared in the objective function. The aircraft balance is violated. So, the value of objective function is sum of the cost of planned aircraft type swapping and cost of aircraft balance disturbance.

Solution 3: The flight 307 (A-P) is realized, after step 1, with 123 minutes delay. In the second part of the step 3, this flight is realized on time by aircraft 14 (instead of 12), while 859 is performed with 91 minutes delay, by aircraft 12 (instead of 14). The objective function represents cost of passengers' delay. Its value is 11.284 units and it is given in Table 2, as is data about flights influenced by disturbance.

Table 2. Data on flights influenced by disturbance, solution 3

	415	416	859	307
Departure airport	A	C	A	A
Arrival airport	C	A	TLV	P
Planned departure time	14:14	17:09	18:25	17:53
Real departure time	14:14	17:09	19:56	17:53
Delay (min)	0	0	91	0
A/C planned to realize the flight	11	11	14	12
A/C which actually realized the flight	12	12	12	14
Number of passengers on flight	137	136	124	145
Value of objective function	124·91= 11.284			

All 6 offered solutions are given in Table 3. In the first and second solution, none of flights is delayed. Solutions 3-6 have one delayed flights. Total delay time goes from 0 to 138 minutes, while number of delayed passengers goes from 0 to 145. The value of objective function varies from 0 to 20.010 units. This DSS is installed on PC Pentium IV and it yields a set of solutions in less than 10 seconds.

Table 3. Data on all offered solutions

Solution list	Number of delayed flights	Total delay (min)	Number of delayed passengers	Number of flights which are not realized according to flight plan	Value of objective function
1	0	0	0	3	0
2	0	0	0	3	3.982,16
3	1	91	124	4	11.284,00
4	1	123	145	2	17.835,00
5	1	127	145	3	18.145,00
6	1	138	145	3	20.010,00

5. Conclusion and future directions

The airline schedule disturbance problem is a problem that airlines face daily. Because of the disturbance, airlines can have additional costs, passengers can be dissatisfied, and the reputation of an airline can be reduced. In order to minimize disturbance effects, the dispatchers in the AOC need to perceive the consequences of different solutions from all aspects, in real time. Decision support systems that can offer several feasible solutions based on airline policy in real time are needed in order to simplify the decision-making process.

This paper presents a mathematical formalization of the airline schedule disturbance problem and a special heuristic algorithm for designing a new daily operational flight schedule due to disturbances. This heuristic algorithm can be used in real time for generating several new daily schedules, which are sorted by decreasing value of the objective function. Any of these flight schedules could be accepted by dispatchers. The proposed model is supported by the corresponding software which is illustrated using a middle size European airline's numerical example.

Based on the review of the developed DSS (i.e. mathematical formalization, the special heuristic algorithm, developed software with user-friendly interface) and numerical examples given in this paper, it can be concluded that: 1) The model yields as a result a list of feasible solutions in a real time, so the decision maker can choose and apply any of the solutions from

the list considering, if necessary, criteria not included into the model. 2) Delaying, cancellation and resource (aircraft) substitution are suggested as the main actions for disturbance problem solving; using spare resource and ferry flights are not foreseen by the developed model. 3) The costs are not presented by real value, but penalties; the dispatcher can change these penalties according to experience, instantaneous traffic situation which has to be solved, or airline policy. New strategies can be defined by changing values of penalties (costs), therefore the DSS can accommodate different airline policies. 4) The models are usually made for specific airline needs; a great advantage of the model presented in this paper is the possibility of changing penalty values which can adapt the model for different airline needs. 5) The developed DSS is easy to use for less experienced dispatchers and a useful tool in dispatcher training. 6) The use of the developed decision support system in the AOC could make the dispatchers' work simpler and faster, as well as foresee the effects of the applied solution.

The model presented in this paper gives solutions that are feasible from the aspect of aircraft availability, while crew availability is not considered despite of the close link between aircraft and crew. In further research these two resources could be merged, so that the obtained solution is feasible from the aspect of both aircraft and crew. This will facilitate the dispatchers' work, because they could reallocate all resources by running single software. This would also additionally reduce the time needed for disturbance problem solving. Ferry flights can be introduced as well as spare aircraft use.

Acknowledgements This paper was written as a part of the author's master thesis. The paper presents extended work on the Airline Schedule Optimization project and Project 15023, sponsored by Serbian Ministry of science and environmental protection.

References

1. Abdelghany, K. F., Abdelghany, A. F., Ekollu, G., An Integrated Decision Support Tool for Airlines Schedule Recovery during Irregular Operation., *European Journal of Operational Research*, 185(2), 825-848, (2008)
2. *Airline Schedule Optimization – Project report*, University of Belgrade, Faculty of Transport and Traffic Engineering, (2004)
3. Andersson, T., Värbrand, P., The Flight Perturbation Problem, *Transportation Planning and Technology*, 27(2), 91-117, (2004)
4. Andersson, T., Solving the Flight Perturbation Problem with Meta Heuristics, *Journal of Heuristics* 12, 37-53, (2006)
5. Bazargan, M., *Airline Operations and Scheduling*, Ashgate, (2004)
6. Jarrah, A. I., Yu, G., Krishnamurthy, N., Rakshit A., A Decision Support Framework for Airline Flight Cancellations and Delays, *Transportation Science* 27(3), 266-280, (1993)
7. Kalić, M., Pavković, G., Operational daily airline scheduling model: Jat Airways example, *Proceedings SYM-OP-IS 2003*, 595-598, (2003)
8. Nedeljković, S., Model for Daily Operational Flight Schedule Designing, *Proceedings ICRAT 2004*, 133-141, (2004)
9. Rakshit, A., Krishnamurthy, N., Yu G., System Operations Advisor: A Real-Time Decision Support System for Managing Airline Operations at United Airlines, *INTERFACES* 26, 50-58, (1996)
10. Rosenberger, J., Johnson, E. Nemhauser, G., Rerouting Aircraft for Airline Recovery, *Transportation Science* 37(4), 408-421, (2003)
11. Stojković, G., *Models for Operational Daily Airline Scheduling*, M. Sc. Thesis, Faculty of Transport and Traffic Engineering, University of Belgrade, (1990) (in Serbian)
12. Stojković, G., Soumis, F., Desrosiers, J., Solomon M., An Optimization Model for a Real-Time Flight Scheduling Problem, *Transportation Research A* 36, 779-788, (2002)
13. Teodorović, D., Stojkovic, G., Model for Operational Daily Airline Scheduling, *Transportation Planning and Technology* 14, 273-286, (1990)

Heuristics for Divisible Loads Scheduling in Systems with Limited Memory

J.Berlińska · M.Drozdowski

Abstract In this paper we study divisible load scheduling in systems with limited memory. Divisible loads represent computations which can be arbitrarily divided into parts and performed independently in parallel. The scheduling problem consists in distributing the load in a heterogeneous system taking into account communication and computation times, and limited memory buffers, so that the whole processing time is as short as possible. The amount of memory available at the processors is too small to accommodate the whole load at once. Thus the load is distributed in many small installments. Since our scheduling problem is computationally hard, we propose a genetic algorithm and several heuristics. We analyze their performance in a series of computational experiments. By comparing quality and cost of the solutions we demonstrate incapability of certain classes of heuristics.

1 Introduction

In divisible load model computations can be divided into pieces of arbitrary sizes, and the pieces can be processed independently in parallel. These two assumptions have practical implications. The grain of parallelism is fine because the load part sizes can be arbitrary. As the parts can be processed independently, there are no precedence constraints between the grains of parallelism. Divisible load model originated in the late 1980s [1, 5]. Surveys of divisible load theory (DLT), including applications, can be found e.g. in [4, 11]. DLT proved to be a valuable tool for modeling processing of big volumes of data.

Distributing the load causes inevitable communication delays. To shorten them, the load may be sent to processors in small chunks rather than in one long message. This way the computations start earlier. Such *multi-installment* or *multi-round* divisible

J.Berlińska
Faculty of Mathematics and Computer Science, Adam Mickiewicz University, Umultowska 87,
61-614 Poznań, Poland
E-mail: Joanna.Berlinska@amu.edu.pl

M.Drozdowski
Institute of Computing Science, Poznań University of Technology, Piotrowo 2, 60-965 Poznań,
Poland
E-mail: Maciej.Drozdowski@cs.put.poznan.pl

load processing was proposed first in [3]. Memory limitations for single-installment communications were studied in [9], where a fast heuristic has been proposed. In [8] it was shown that this problem is **NP**-hard if a fixed startup time is required for initiation of communications. Divisible load processing with both multiple rounds and memory constraints has been studied in [6]. However, the memory management model used in [6] was simplifying reality. We discuss it in more detail in the next section.

In this paper we study multi-round divisible load scheduling in heterogeneous system, with communication startup time, memory constraints and arbitrary communication sequence. As the problem is known to be computationally hard, we propose several types of heuristics. In a series of computational experiments we test quality and cost of the proposed algorithms, their scalability and susceptibility to system heterogeneity.

The rest of this paper is organized as follows. In Section 2 the problem is formulated. Section 3 describes algorithms for solving our problem. The results of computational experiments are presented in Section 4. The last section is dedicated to conclusions.

2 Problem Formulation

In this paper we assume star interconnection. A set of working processors $\{P_1, \dots, P_m\}$ is connected to a central server P_0 called originator. A processor is a unit comprising a CPU, memory and a hardware network interface. The CPU and network interface can work in parallel so that simultaneous communication and computation is possible. A processor P_i , $i = 1, \dots, m$, has computing rate A_i , size of available memory B_i , communication rate C_i and communication startup time S_i . The time required to transfer α units of load from P_0 to P_i is $S_i + \alpha C_i$, and the time needed to process this load is αA_i . Initially the originator has some volume V of load to process. The problem consists in finding a communication sequence, i.e. the schedule of communications from the originator to the workers, and sizes of transferred load pieces. A star topology may represent a cluster of workstations connected via a local area network, a set of CPUs in a symmetric multiprocessor (SMP) system sharing a bus, as well as distributed computation in a grid environment.

Now let us consider memory management. The simplest approach, used in [8,9], is to assume that only one load chunk is held by a processor at a time. However, this method is insufficient for multi-installment processing, because many messages can arrive at the same processor. In [6] it was assumed that the whole received load *together* should not exceed memory size B_i at any moment. However, it was also assumed in [6] that memory is released with very fine granularity equal to one load unit, so that memory occupation decreases linearly during computations. This way of releasing memory is rather unusual because releasing memory in many fine size pieces would also require this many memory allocations. Obtaining memory chunk would require multiple malloc/new calls to the runtime environment. Consequently, the process of acquiring memory would be unnecessarily complicated, and certainly not instantaneous. In this paper we use a realistic memory management where memory allocation and release have block nature. When a chunk of size α_j is about to arrive at a processor, a block of size α_j is requested from the operating system. It exists in the memory pool of the application until finishing computation on this chunk. The sizes of coexisting memory blocks cannot exceed the limit B_i . In other words, for each moment t , $\sum_{l \in \mathcal{H}(i,t)} \alpha_l \leq B_i$, where $\mathcal{H}(i,t)$ is the set of chunks received by P_i and not completed by time t . We will be saying that chunks simultaneously existing in the memory buffer *overlap*.

Our scheduling problem has been formulated as a mixed nonlinear mathematic program in [7]. Here we will formulate this problem in a simpler way by dividing the solution into two parts: combinatorial and algebraic. Let us introduce necessary assumptions and notations. The load is sent to the processors in a sequence of communications. If the load is received by a processor with an empty buffer, then computations start immediately after the end of communication. If the buffer stores some unprocessed chunks, then computations on consecutive chunks are performed without delay, in the order of receiving the chunks. Let us assume that the communication sequence $\sigma = (\sigma(1), \dots, \sigma(n))$ is given, where $\sigma(i)$ is the number of the processor receiving the i -th message. The numbers of the chunks as they are sent off the originator will be called global numbers. Hence, i in $\sigma(i)$ is a global number. For simplicity of notation we will also use a local numbering of chunks received by a given processor. Function $\rho(i, j)$ is a mapping from processor P_i local chunk number j to the global numbering. For example, if the k -th chunk sent off the originator is received as the j -th chunk on processor P_i , then the global number of this chunk is k , the local number of this chunk is j and $\rho(i, j) = k$. Assume that z_{ij} denotes the local number of the last chunk overlapping with chunk j (local number) on processor P_i . Let n_i denote the number of chunks received by P_i . We will denote the moment when sending of the i -th chunk starts by t_i (i is a global chunk number here). The moment when computation on chunk k (local number) of P_i finishes will be denoted by f_{ik} . Our scheduling problem can be formulated in the following way:

minimize T_{max}
subject to LP:

$$t_1 = 0 \quad (1)$$

$$t_i \geq t_{i-1} + S_{\sigma(i-1)} + C_{\sigma(i-1)}\alpha_{i-1} \quad i = 2, \dots, n, \quad (2)$$

$$f_{ik} \geq t_{\rho(i,k)} + S_i + C_i\alpha_{\rho(i,k)} + A_i\alpha_{\rho(i,k)} \quad (3)$$

$$i = 1, \dots, m, \quad k = 1, \dots, n_i,$$

$$f_{ik} \geq f_{i,k-1} + A_i\alpha_{\rho(i,k)} \quad (4)$$

$$i = 1, \dots, m, \quad k = 2, \dots, n_i,$$

$$f_{ij} \geq t_{\rho(i,z_{ij})} \quad i = 1, \dots, m, \quad j = 1, \dots, n_i - 1 \quad (5)$$

$$f_{ij} < t_{\rho(i,z_{ij}+1)} \quad i = 1, \dots, m, \quad j = 1, \dots, n_i - 1 \quad (6)$$

$$\sum_{k=j}^{z_{ij}} \alpha_{\rho(i,k)} \leq B_i \quad i = 1, \dots, m, \quad j = 1, \dots, n_i \quad (7)$$

$$V = \sum_{i=1}^n \alpha_i \quad (8)$$

$$T_{max} \geq f_{in_i} \quad i = 1, \dots, m \quad (9)$$

In the above formulation variables α_i define chunk sizes resulting in minimum schedule length T_{max} for communication sequence σ and overlapping defined by values z_{ij} . Constraints (1), (2) determine the moment when sending of chunk i starts. Inequalities (3),(4) determine the earliest moment when computation on the k -th chunk sent to P_i may finish. By inequality (5) processing of chunk j is not finished before starting message z_{ij} , and by inequality (6) it is finished before chunk $z_{ij} + 1$ starts arriving. Inequalities (7) guarantee that memory limits are observed. All load is processed by

(8). By constraints (9) schedule length is not shorter than the completion time on any processor. Formulation (1)-(9) is a linear program for a given communication sequence σ and the depth of overlap z_{ij} . Calculating the optimum values of α_i , t_i , f_{ik} , T_{max} from (1)-(9) constitutes the algebraic part of the problem. The combinatorial part is finding optimum communication sequence σ and values z_{ij} . Splitting our problem into such two parts is a foundation of some algorithms proposed in the next section.

3 Algorithms

In the previous section we showed that our problem can be solved by a sequence of two methods. The first one solves the combinatorial part by choosing communication sequence σ and overlap z_{ij} , and the second solves the LP, computing chunk sizes α_i .

In the actual algorithms we used a more convenient encoding of the chunk overlap. Instead of z_{ij} s we used variables δ_{ij} denoting by how many chunks z_{ij} progressed from $z_{i,j-1}$. For the given values of δ_{ij} we can compute values $z_{ij} = \min\{n_i, \max\{z_{i,j-1}, j\} + \delta_{ij}\}$, where $z_{i0} = 1$. This overlap encoding is used in algorithms presented in the following sections. For given σ and δ_{ij} distribution of the load can be obtained from the linear program (1)-(9).

3.1 Genetic Algorithm

Our first approach to solve the analyzed scheduling problem was to construct a Branch and Bound algorithm (BB). This is described in detail in [2]. It turned out that BB is practically unusable for solving even moderate size instances of our problem due to huge execution times. Therefore, we had to construct a faster algorithm which could be used as a reference for the proposed heuristics.

Genetic algorithm (GA) is a randomized search method using a set of operators to combine pieces of good solutions. We present here only basics of our GA implementation (see [2] for details).

The solutions are encoded as pairs of strings: the communication sequence σ and the overlap sequence O (which comprises δ_{ij} s). More precisely, $O(i) = \delta_{\sigma(i)j}$, where j is the number of chunks received by processor $P_{\sigma(i)}$ up to the i -th chunk (global number) sent off the originator. The lengths of σ and O are equal and can be adjusted by GA. Optimum chunk sizes α_i are computed by a linear program defined in the previous section. Fitness of a solution is measured as the inverse of schedule length T_{max} obtained from the linear program for given σ and O .

Our population is a set of G solutions. Three genetic operators were used: crossover, mutation and selection. Two-point crossover was applied. Crossover, in general, changes the length of the chromosome. The number of the crossover offspring is Gp_C , where p_C is a tunable parameter called crossover probability. Mutation changes $E(t)p_M$ random genes, where $E(t)$ is the total number of genes in the population in generation t and p_M is tunable mutation probability. In the selection operation a new population is assembled. The best $G(1 - p_C)$ solutions are always preserved and the rest of chromosomes are crossover offspring. The algorithm stops after a fixed number of iterations it_1 . There is also another limit it_2 for iterations without improvement in solutions quality. If this limit is reached before it_1 , the population is replaced with randomly

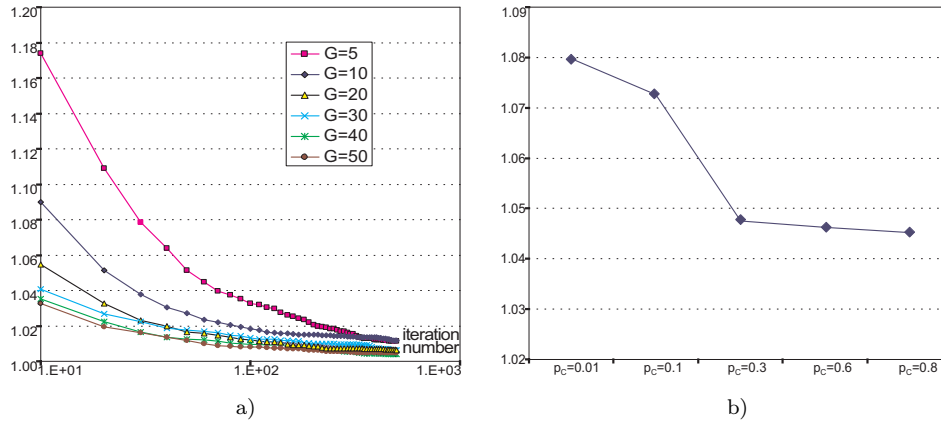


Fig. 1 GA tuning. a) Solution quality vs. population size G , b) solution quality at 100th iteration vs. p_C .

generated solutions and the search is restarted. The best solution found so far is always recorded.

GA parameters must be tuned. We applied the following procedure. A set of 200 random instances with $m = 3, \dots, 6$, $V = 20$, B_i uniformly distributed in $[0, 10]$, A_i, C_i, S_i uniformly distributed in $[0, 1]$, were generated and solved to the optimum by BB. Due to relatively large values of B_i the optimum communication sequences were short enough to calculate them by BB algorithm. The average relative distance of the schedule length T_{max} from the optimum length was the measure of the tuning quality. The tunable parameters were selected one by one. The process of selecting the tunable parameters is illustrated in Fig.1. The population size $G = 20$ was selected as a compromise between the speed of convergence to the near-optimum solutions and the computational complexity (cf. Fig.1a). To select the crossover probability, mutation operator was switched off. Crossover probability $p_C = 0.8$ was selected (Fig.1b). Thus, majority of the population (80%) are offspring. After fixing G and p_C , mutation probability $p_M = 0.1$ was chosen. In the end, values $it_1 = 100$, $it_2 = 10$ were selected as a compromise between quality and complexity. We skip the details of selecting p_M , it_1 , it_2 , which can be found in [2].

3.2 Random Heuristics

The random heuristics were designed mainly to verify the performance of other heuristics. Heuristic **Rnd1** appends random processors to communication sequence σ until the sum of their memory limits is not less than load size V . Each chunk carries load equal to the processor memory limit. Therefore, chunks do not overlap.

The other random heuristic, **Rnd2**, not only appends random processors to communication sequence σ , but also chooses chunk sizes randomly between 0 and memory limit size for a given processor. If the size of free memory at the chosen receiver processor is smaller than the chosen chunk size, then sending the chunk is delayed until enough memory is released. This process is repeated until all the load is sent.

3.3 Appender heuristics

For the purpose of constructing communication sequence σ it is assumed in all appender heuristics that each chunk has (phantom) size equal to the memory limit. Load is always sent to the first free processor. The processors are checked in the order specified by a combination of their parameters. Processors are appended to communication sequence until the (phantom) load sent in all communications is at least three times greater than V . The aim of creating such a long communication sequence is to give some freedom in choosing chunk sizes. Finally, overlap $\delta_{ij} = 1$ is applied to all chunks (thus each two consecutive chunks overlap) and optimal chunk sizes are computed by LP (1)-(9).

In appender heuristics **apA**, **apC**, **apS** and **apB** a chunk is sent to a free processor with the best value of a given single parameter (i.e. smallest A_i , C_i , S_i and greatest B_i respectively). Heuristic **apSBCr** searches for a processor with the best communication rate, i.e. the one with the smallest value of $S_i/B_i + C_i$. In heuristic **apSBCAr** the processors are served in the order of increasing values of $S_i/B_i + C_i + A_i$. Heuristics with other combinations of processor parameters were also examined, but their performance was similar to the above ones. Therefore, we do not present them.

3.4 Best Processing Rate Heuristics

In best processing rate heuristics **BRx**, it is assumed that the size of each chunk sent to processor P_i is equal to $\beta_i = B_i/x$. Thus, each chunk size is a constant part of the memory size of a receiver. Using several equal size buffers simplifies a parallel application. As memory limits must be observed, x cannot be less than 1. Sending very small chunks is not effective, because the number of communication startups gets very large. Therefore, we decided to use $x \in \{1, \dots, 4\}$. To choose a receiver of a chunk, we compute the time T_i needed to process the chunk on each processor P_i :

$$T_i = \max\{\max\{t_0, \tau_i\} + S_i + \beta_i(C_i + A_i), t_i + \beta_i A_i\} - t_0,$$

where t_0 is the time when the originator can start sending the chunk, τ_i is the time when enough memory becomes available at processor P_i , and t_i is the first moment when processor P_i completes processing the preceding chunks and can start processing this chunk. The load is sent to a processor with the minimum value of T_i/β_i . The process is repeated until all load is sent. The construction of BRx algorithms prevents using more memory than available. As chunk sizes are equal to B_i/x , at most x chunks may overlap any time on any processor.

4 Computational Experiments

As the worst-case estimates of the quality of the proposed algorithms are often pessimistic, we decided to examine their performance experimentally. Since our model can be applied in a wide range of diverse computing environments and no generally representative real workloads are known, we used random instances.

In our experiments instance parameters A , B , C , S were generated randomly from range $(0, 1]$ and the number of processors from $[1, 10]$. Experiments were repeated for $V \in \{2, 5, 10, 20, 50\}$. The schedules consisted of up to 1000 messages. Over 15000 instances were solved by the analyzed algorithms. The experiments were run on clusters of 15-75 PCs with Linux. Linear programs were solved using **lp_solve** package [10].

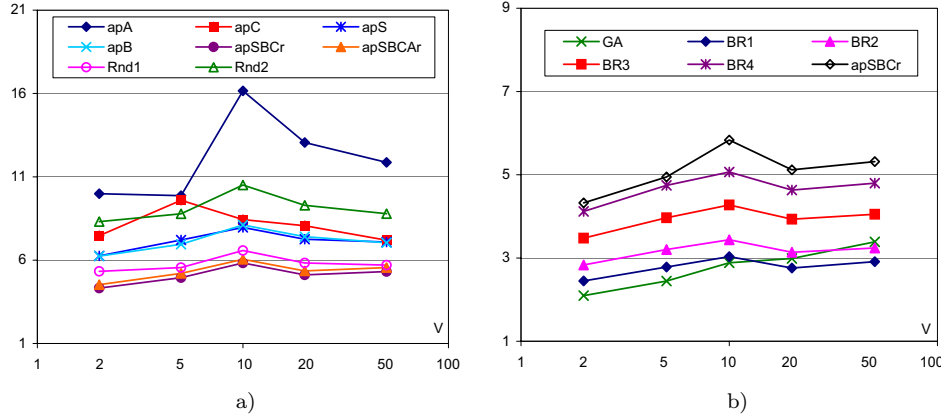


Fig. 2 Average solution quality vs. V . a) random and appender heuristics, b) GA, best rate heuristics and apSBCr.

4.1 Quality of the Solutions

We measured the quality of all algorithms as the relative distance from the lower bound (LoBo) calculated as follows. The minimum communication time is $\tau_1 = n_{MIN}S_{min} + VC_{min}$, where $n_{MIN} = \lceil \frac{V}{\max_{i=1}^m B_i} \rceil$ is the minimum required number of chunks, $C_{min} = \min_{i=1}^m C_i$, $S_{min} = \min_{i=1}^m S_i$. At most $V_0 = (\tau_1 - S_{min}) \sum_{i=1}^m \frac{1}{A_i}$ load could be processed in parallel with the communication time τ_1 . If any load is remaining, i.e. $V - V_0 > 0$, then it has to be processed in time at least equal to $\frac{V - V_0}{\sum_{i=1}^m 1/A_i}$. Thus, the lower bound is equal to $LoBo = \tau_1 + \max\{0, \frac{V}{\sum_{i=1}^m 1/A_i} - \tau_1 + S_{min}\}$.

The quality of the obtained solutions for different values of V is shown in Fig.2. Let us start with comparing appender and random heuristics (Fig.2a). It can be seen that only apSBCr and apSBCAr perform better than heuristic Rnd1. Appender apA is even worse than heuristic Rnd2, which creates completely random schedules. We infer that appenders are not a good tool for solving our problem. Better results are obtained taking into account a combination of processor parameters instead of a single parameter.

In Fig.2b the results of BRx heuristics, GA, and the best appender are shown. The best of BRx heuristics is BR1; growing x causes worse results. However, even BR4 is better than the best appender. GA delivers the best solutions for smaller load, but at $V = 20$ heuristic BR1 is better, and for $V = 50$ even BR2 obtains better solutions than GA. This may be explained by the fact that for growing V longer communication sequences are needed. The search space of GA becomes too big and the search efficiency decreases. Instances with bigger load are more difficult for all algorithms (cf. Fig.2a,b), but this effect is strongest for GA. Thus, GA does not scale well with growing problem size V .

The comparison of average quality and execution times of all presented algorithms for all test instances described above is shown in Fig.3. Note that this is a new way of depicting performance of heuristics. The points on this chart show average solution quality against average computational cost. Thus, in Fig.3 heuristics are shown as

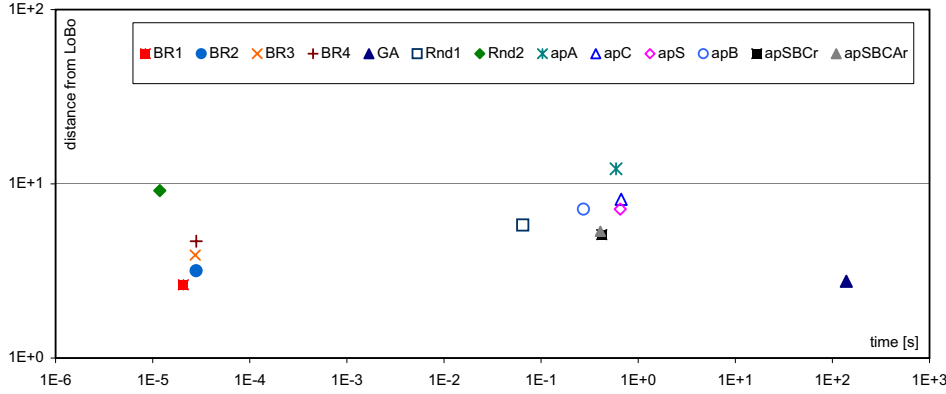


Fig. 3 Solution quality vs. execution time of all algorithms.

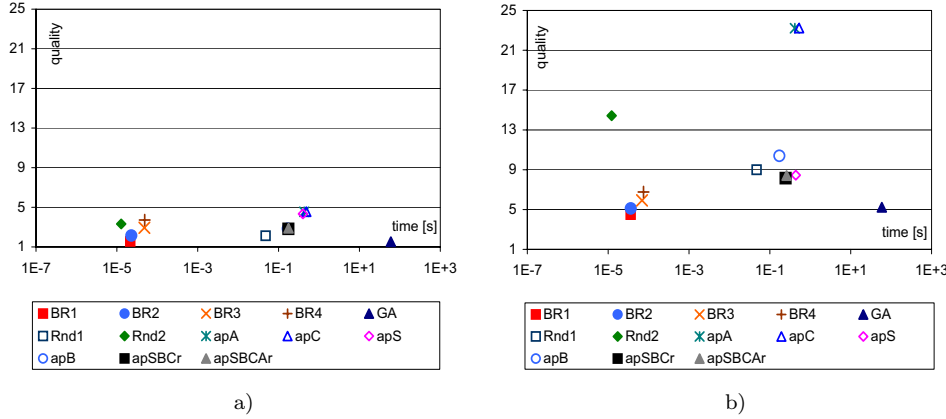


Fig. 4 Solution quality vs. execution time. a) $\Delta_S = 0.001$, b) $\Delta_S = 1$.

objects with two attributes: quality (along the vertical axis), and computational cost (horizontal axis). In Fig.3 the algorithms can be divided into three groups taking into account their execution times. The genetic algorithm is the slowest as it generates many solutions while searching for the best one. The middle group are heuristics using linear programming. The BRx heuristics and Rnd2, which do not use LP, have the shortest processing time.

In the time vs. quality trade-off Rnd2, BR1, GA seem Pareto-optimal. On average, the quality of BR1 and GA solutions is very close. Moreover, the running time of BR1 is very short (only Rnd2 is faster). Thus, for random, non-biased instances BR1 seems to be the best algorithm.

4.2 Influence of Heterogeneity

The second set of experiments examined the influence of system heterogeneity on the obtained results. We tested the influence of dispersion of parameters A , C , S , B/V . In

the experiments concerning a given parameter (say S) this parameter was generated randomly from $(0.5 - \frac{\Delta_S}{2}, 0.5 + \frac{\Delta_S}{2}]$, where Δ_S was a fixed value. The remaining instance parameters were generated as described above. We present only the results for the extreme values $\Delta_S = 0.001$ and $\Delta_S = 1$ (cf. Fig.4). All the results get much worse when parameter S is chosen from a wider range ($\Delta_S = 1$). This means that our problem, by its nature, is more difficult in a more heterogeneous system. A similar effect was observed for the dispersion of the other parameters.

5 Conclusions

In this paper we proposed methods for scheduling divisible computations on a heterogeneous system with limited memory. As the problem is computationally hard, we designed a genetic algorithm and several heuristics. The computational experiments allowed for choosing heuristic BR1 as the best of all proposed algorithms, because it provides good solutions in relatively short time. Appender heuristics appear to be inefficient both in quality and execution time. GA has high computational cost and does not scale well with the problem size V . We showed that growing computer system heterogeneity makes the analyzed problem more difficult for all presented algorithms. Future research may include analyzing the influence of particular system parameters on the performance of the proposed algorithms.

Acknowledgements This research was partially supported by a grant of Polish Ministry of Science and Higher Education N N519 1889 33.

References

1. R. Agrawal, H.V. Jagadish, Partitioning Techniques for Large-Grained Parallelism, IEEE Transactions on Computers 37(12), 1627-1634 (1988)
2. J. Berlińska, M. Drozdowski, M. Lawenda, Multi-Installment Divisible Loads Scheduling in Systems with Limited Memory, Research Report RA-07/08, Institute of Computing Science, Poznań University of Technology (2008). <http://www.cs.put.poznan.pl/mdrozdowski/rapIIIn/ra0708.pdf>
3. V.Bharadwaj, D.Ghose, V.Mani, Multi-installment Load Distribution in Tree Networks with Delays. IEEE Transactions on Aerospace and Electronic Systems 31, 555-567 (1995)
4. V.Bharadwaj, D.Ghose, V.Mani, T.G.Robertazzi. Scheduling Divisible Loads in Parallel and Distributed Systems, IEEE Computer Society Press, Los Alamitos CA (1996)
5. Y.-C.Cheng, T.G.Robertazzi, Distributed computation with communication delay. IEEE Transactions on Aerospace and Electronic Systems 24, 700-712 (1988)
6. M.Drozdowski, M.Lawenda, Multi-installment Divisible Load Processing in Heterogeneous Systems with Limited Memory, in: R.Wyrzykowski et al., PPAM 2005, Lecture Notes in Computer Science 3911, 847-854 (2006)
7. M. Drozdowski, M. Lawenda, A New Model of Multi-installment Divisible Loads Processing in Systems with Limited Memory, in: R.Wyrzykowski et al., PPAM 2007, Lecture Notes in Computer Science 4967, 1009-1018 (2008)
8. M. Drozdowski, P. Wolniewicz, Optimum divisible load scheduling on heterogeneous stars with limited memory, European Journal of Operational Research 172, 545-559 (2006)
9. X.Li, V.Bharadwaj, C.C.Ko, Processing divisible loads on single-level tree networks with buffer constraints, IEEE Transactions on Aerospace and Electronic Systems 36, 1298-1308 (2000)
10. Lp_solve reference guide, <http://lpsolve.sourceforge.net/5.5/> (2007)
11. T.G. Robertazzi. Ten reasons to use divisible load theory, IEEE Computer, 36, 63-68 (2003)

Three Heuristic Procedures for the Stochastic Flow Shop Problem

Kenneth R. Baker • Dan Trietsch

Abstract. Although the deterministic flow shop model is one of the most widely studied problems in scheduling theory, its stochastic analog has remained challenging. No computationally efficient optimization procedure exists even for the general two-machine version. In this paper, we describe three heuristic procedures for the stochastic two-machine flow shop problem and report on computational experiments to compare their effectiveness. We find that all three are reliably better than using an arbitrary sequence, and they are capable of generating solutions close to optimal. In addition, the conditions under which they perform well can vary, and our experiments shed some light on these differences.

1 Introduction

The analysis of stochastic flow shop problems has not penetrated very far and remains challenging. With few exceptions, analytic results for the stochastic flow shop have been limited to the expected completion time of all jobs—the *makespan*—as a performance measure, and much of the work addresses only the two-machine problem. In the deterministic case, the minimization of the makespan for the two-machine model is well solved, and a vast literature exists on multi-machine models and on other performance measures. In the stochastic case, the two-machine makespan problem is inherently more complex than its deterministic counterpart. This paper presents a computational comparison of three heuristic procedures for minimizing the expected makespan in the stochastic case.

After introducing the flow shop model, we review the few theoretical results that exist for the stochastic case and thereby lay the groundwork for the heuristic procedures that are the focus of the study. Then we describe our computational experiments and summarize our findings. Finally, we suggest extensions of this work that would improve our knowledge about solving stochastic flow shop problems.

In the two-machine flow shop, we face the problem of scheduling n jobs that are simultaneously available at time zero. Each job requires processing first at Machine A and then at Machine B. Processing on Machine B cannot begin on a job until its processing on Machine A has completed and the preceding job has completed its processing on Machine B. For job j , we use a_j and b_j to denote processing times on Machines A and B, respectively. Assuming for the moment that the jobs are processed in numbered order, we have for job j :

$$\text{Completion time on Machine A: } C_{ja} = \sum_{i=1}^j a_i$$

$$\text{Completion time on Machine B: } C_{jb} = \max\{C_{ja}, C_{j-1,b}\} + b_j$$

Kenneth R. Baker
Dartmouth College
ken.baker@dartmouth.edu

Dan Trietsch
American University of Armenia
dan.trietsch@gmail.com

The makespan is then $C_{nb} = \max_j \{C_{jb}\}$.

When the processing times are known, the optimal schedule is a permutation schedule (in which the same sequence occurs on both machines), and the optimal job sequence may be determined by Johnson's Rule [1]: job i precedes job j in an optimal sequence if $\min\{a_i, b_j\} \leq \min\{a_j, b_i\}$. This condition can be translated into an efficient sequencing algorithm that is computationally equivalent to sorting the jobs.

2 Stochastic counterpart models

In the stochastic case, we use A_j and B_j to denote (random) processing times on the two machines, but we retain a_j and b_j to represent their expected values. We assume that the probability distributions of the processing times are known and that the processing times are independent random variables. (Usually, the problem is specified by assuming a particular family of processing times, such as the normal or the exponential distribution, and then specifying the parameters of that family as they apply to each processing time.) The objective in the stochastic problem is to minimize the expected value of the makespan. We refer to this problem as the *stochastic counterpart* of the two-machine flow shop problem.

As in the deterministic case, we can limit our attention to permutation schedules [2]. However, for general distributions without special conditions on processing times, only one full solution is known for the stochastic counterpart, given by Makino [3].

Theorem 1 In the two-job stochastic flow shop problem, job 1 precedes job 2 in an optimal sequence if $E(\min\{A_1, B_2\}) \leq E(\min\{A_2, B_1\})$.

Theorem 1 generalizes Johnson's Rule for two jobs in the deterministic case. Although the theorem holds for any joint processing time distributions, the calculations become more tractable if we assume independent processing time distributions. With independent distributions, we can express the cumulative distribution function (cdf) of the minimum of two random variables X and Y by using the formula

$$F_{\min}(t) = 1 - (1 - F_X(t))(1 - F_Y(t)), \quad (1)$$

where $F_X(t)$, $F_Y(t)$ and $F_{\min}(t)$ denote the cdfs of X , Y and their minimum. However, this formula does not lead to a closed-form calculation of the minimum for all distributions, so Theorem 1 is not always easy to implement in the general case.

To make further progress with the analysis, we must make a more specific assumption about the distribution of processing times. To start, we consider exponential processing times. (The exponential is not necessarily very practical as a model for job processing times, but it is significant as a boundary case, and it sometimes provides us with general insights.) Suppose we have two exponential random variables, one with a mean of a and the other with a mean of b . Equivalently, we say that one has a *processing rate* (or *completion rate*) of $1/a$ and the other has a rate of $1/b$. The minimum of the two exponential random variables is an exponential random variable with processing rate of $(1/a + 1/b)$. Therefore, the condition in Theorem 1 can be rewritten for the exponential case. The condition for job 1 to precede job 2,

$$E(\min\{A_1, B_2\}) \leq E(\min\{A_2, B_1\}), \quad (2)$$

implies the reverse ordering of processing rates, or

$$1/a_1 + 1/b_2 \geq 1/a_2 + 1/b_1 \quad (3)$$

Algebraic rearrangement yields:

$$1/a_1 - 1/b_1 \geq 1/a_2 - 1/b_2 \quad (4)$$

In words, the job with the larger difference in processing rates should come first.

Example 1 Consider the scheduling of two jobs with independent exponentially-distributed processing times. The mean of each processing time is given in the following table.

Machine	Job	1	2
1		4	5
2		5	8

The processing rates associated with these means are 0.25 and 0.2 for job 1 and 0.2 and 0.125 for job 2. Since the difference for job 2 (0.0875) exceeds the difference for job 1 (0.05), job 2 should be scheduled first. This sequence reverses the solution of the deterministic counterpart and demonstrates that in a stochastic problem, applying Johnson's Rule to the mean values does not necessarily produce an optimal sequence. For this example, the expected makespan is 19.333 for the sequence (2-1) and 19.5 for (1-2). The standard deviations are 10.424 and 10.548, respectively. The difference in both measures is on the order of 1%, and the optimal sequence is advantageous on both counts.

Example 2 Consider the scheduling of two jobs with independent exponentially-distributed processing times. The mean of each processing time is given in the following table.

Machine	Job	1	2
1		1	1
2		1	$1+\sqrt{7}$

In Example 2, the calculations yield an expected makespan of 5.861 for the sequence (2-1) and 6.146 for (1-2). The standard deviations are 3.801 and 3.942, respectively. For the optimal sequence, the advantage in the mean is about 4.8%, and the advantage in the standard deviation is about 3.7%. The parameters of this example illustrate the maximum percentage improvement of the correct sorting rule for a two-job problem with exponential processing times. Johnson's Rule, when applied to the means, results in a tie, but if we reduce a_1 infinitesimally, the sequence (1-2) becomes the deterministic optimum, whereas the sequence (2-1) remains the optimal solution.

Sorting by the difference in mean processing rates ($1/a_j - 1/b_j$) involves a transitive sequencing relation and can be extended to n jobs. In the exponential case, this sorting rule, known as Talwar's Rule, also turns out to be optimal. Talwar [4] originally conjectured optimality, although the first proof was obtained by Cunningham and Dutta [5]. Ku and Niu [2] showed that Talwar's rule is not only optimal for the stochastic case but also yields a stochastically dominant makespan.

Theorem 2 (Talwar's Rule) In the stochastic two-machine flow shop problem with exponential processing times, the expected makespan is minimized stochastically by sequencing the jobs in nonincreasing order of $(1/a_j - 1/b_j)$.

To summarize the state of the art in stochastic flow shop problems, we have two theoretical results, namely Theorems 1 and 2. Theorem 1 is a distribution-free result, but it holds only for the two-job problem. Theorem 2 holds for the exponential distribution and provides us with a transitive rule for sequencing the jobs optimally.¹ If we seek practical methods for solving a stochastic flow shop problem in which the processing times are not exponential, we shall have to develop a heuristic scheduling procedure.

3 Heuristic procedures

When the problem involves distributions other than the exponential, we have to rely on heuristic procedures because no result as strong as Theorem 2 has been developed for any other family of processing time distributions. One plausible heuristic is to follow Talwar's Rule and sort the jobs by nonincreasing differences in processing rates. This approach makes sense if we

¹ Nevertheless, the computational effort required to calculate the expected makespan, even when we know the optimal sequence, grows exponentially with the problem size. Even in the case of exponential distributions, where the memoryless property facilitates the computations, the number of subcases to be evaluated is of the order 2^{n-2} .

believe that the analysis of the exponential case captures relevant information about the problem in general. (As mentioned earlier, however, the exponential is a boundary case. It is a one-parameter distribution with standard deviation equal to its mean and is therefore not necessarily a good model for the kinds of stochastic behavior found in practice.) Thus, our first heuristic solution procedure might be called *Talwar's Heuristic*: sequence the jobs in nonincreasing order of the difference between their two processing rates.

Another heuristic, of course, is to solve the deterministic counterpart. This approach suppresses the randomness in the problem and replaces the stochastic processing times with deterministic equivalents. A general method for solving a stochastic scheduling problem is to replace its stochastic variables by their means and proceed with deterministic analysis to find a sequence. This approach is commonly found in practice, especially in applications where data are scarce and distribution information is difficult to obtain. We refer to this heuristic procedure as *Johnson's Heuristic*. Portugal and Trietsch [6] showed that Johnson's Heuristic is asymptotically optimal under mild regularity conditions—that is, as n grows large, the relative difference between the heuristic makespan and the optimal makespan tends to zero.

In a sense, both Talwar's Heuristic and Johnson's Heuristic presume that the mean of the distribution represents all the information we need to know about the probability model in order to make a sound decision about job sequence. Neither approach recognizes variability apart from the estimate of the mean in order to influence job sequence. By contrast, a heuristic procedure that looks beyond mean-value information seems desirable. Such an approach, based on the properties of an adjacent pairwise interchange (API), has been developed by Portugal and Trietsch. Their API procedure starts with the Johnson sequence and employs an API neighborhood search. In principle, however, any other sequence could be used as the basis of the search. We call a pair of adjacent jobs *stable* if they satisfy Theorem 1. We call a sequence *stable* if every pair of adjacent jobs is stable. The API procedure checks all adjacent job pairs in the initial sequence, and if an unstable pair is found, the two jobs are interchanged. After the interchange, the upstream (earlier) job may not form a stable pair with its new upstream neighbor, in which case we interchange those jobs, too. After each interchange, the job that was moved downstream is stable with respect to its upstream neighbor, but the one moved upstream may have to be interchanged repeatedly until it reaches a stable position, possibly as the first job. Then, the heuristic resumes its check of all pairs and stops when all adjacent pairs are stable.

The API procedure requires $O(n^2)$ tests and always results in a stable sequence. However, the optimal sequence may not be stable, and more than one stable sequence can exist (see Baker and Trietsch [7]). The heuristic is guaranteed only to converge to *some* stable sequence, but that does not mean it will locate the best one. However, if any transitive sorting rule is optimal then the API procedure will converge to it. For example, it would yield the Talwar sequence for exponential distributions.

The key step in the API procedure tests Theorem 1 as it applies to any pair of adjacent jobs. Equivalently, the condition in (2) must be evaluated. However, as we pointed out earlier, the formulas in that evaluation might be intractable for many types of probability distributions. Therefore, we can approximate the desired evaluation by assuming a particular family of distributions for all processing times. If we were to assume an exponential distribution for each processing time, we would generate the sequence of Talwar's Heuristic. Instead, we adopt a more flexible approach and evaluate (2) by assuming the form of a normal distribution. The normal assumption is a two-parameter distribution, unlike the exponential, so we can match its mean and standard deviation with those of the distributions in the problem.

Next, we describe the special-case calculation for the normal distribution. In general, for any random variables X and Y ,

$$\min\{X, Y\} = X - \max\{X - Y, 0\}$$

and

$$E[\min\{X, Y\}] = E[X] - E[\max\{X - Y, 0\}].$$

Define $W = X - Y$ so that

$$E[\min\{X, Y\}] = E[X] - E[W^+].$$

Suppose that X and Y are independent normal random variables. Then, W is normal with mean $\mu_{xy} = \mu_x - \mu_y$ and variance $\sigma_{xy}^2 = \sigma_x^2 + \sigma_y^2$. In addition, $E[W^+] = \sigma_{xy}[\phi(z_{xy}) + z_{xy}\Phi(z_{xy})]$, where ϕ and

Φ denote the density function and the cumulative distribution function of the standard normal and $z_{xy} = \mu_{xy}/\sigma_{xy}$. The expected minimum (see [7], pp. 468-69 for details) is then given by

$$E[\min\{X, Y\}] = \mu_x - \sigma_{xy}[\phi(z_{xy}) + z_{xy}\Phi(z_{xy})] \quad (5)$$

Therefore, the condition for job 1 to precede job 2 takes the following form.

$$\mu_{x1} - \sigma_{x1y2}[\phi(z_{x1y2}) + z_{x1y2}\Phi(z_{x1y2})] \leq \mu_{x2} - \sigma_{x2y1}[\phi(z_{x2y1}) + z_{x2y1}\Phi(z_{x2y1})]$$

This form of the API procedure uses the properties of normal distributions, but it can be used as a heuristic when the distributions are non-normal, as long as we know their means and variances. Implementing the API procedure based on the calculations in (5) gives rise to what we call the *API Heuristic*.

Example 3 Consider the six-job, two-machine flow shop problem shown in Table 1, in which the jobs have normally-distributed processing times on the two machines, A and B.

Table 1

Job	1	2	3	4	5	6
mean A	12.3	17.2	17.6	13.9	16.4	12.7
s.d. A	1.6	1.6	1.2	2.0	2.4	1.5
mean B	14.7	13.1	17.6	12.2	14.1	12.3
s.d. B	1.7	1.4	1.6	2.5	1.4	2.1

We illustrate the algorithm by starting with the sequence 1-2-3-4-5-6. Using (5), Table 2 records the differences between $E[\min\{A_i, B_j\}]$ and $E[\min\{A_j, B_i\}]$ for these jobs, with job i corresponding to a row and j to a column. For instance, the calculations for $E[\min\{A_1, B_2\}]$ yield $\mu_{x1} = 12.3$, $\sigma_{x1y2}^2 = 1.6^2 + 1.4^2 = 2.13^2$, so $z_{x1y2} = (12.3 - 13.1)/2.13 = -0.3763$, leading to $E[\min\{A_1, B_2\}] = 12.3 - 2.13[0.3717 + (-0.3763) \times 0.3534] = 11.792$. Similarly, $E[\min\{A_2, B_1\}] = 14.530$. The first entry in the table is the difference $11.792 - 14.530 = -2.74$. For the algorithm, we need to know only the sign of this calculation. In this case, the negative value indicates that the first two jobs are stable in the sequence 1-2.

Table 2

	1	2	3	4	5	6
1		-2.74	-2.33	-2.14	-2.12	-1.22
2	2.74		3.39	-0.33	1.09	0.20
3	2.33	-3.39		-1.64	-1.67	-0.40
4	2.14	0.33	1.64		1.01	0.50
5	2.12	-1.09	1.67	-1.01		-0.25
6	1.22	-0.20	0.40	-0.50	0.25	

Suppose now that we start arbitrarily with the sequence 1-2-3-4-5-6. As indicated above, jobs 1 and 2 are stable in this sequence. However, jobs 2 and 3 are not, so we interchange them, observing that jobs 1 and 3 are stable, so we obtain the sequence 1-3-2-4-5-6. Next, jobs 4 and 5 are not stable, so we interchange them and then interchange jobs 2 and 5, yielding the sequence 1-3-5-2-4-6. Next, jobs 4 and 6 are not stable, so we interchange them and then interchange jobs 2 and 6 yielding the stable sequence 1-3-5-6-2-4.

For the parameters of Example 3, Johnson's Heuristic yields the sequence 1-3-5-2-6-4, and Talwar's Heuristic yields the sequence 1-3-6-5-4-2. Thus, the three heuristics produce three different sequences. In other instances, they may produce two different sequences or they may all produce the same sequence.

4 Computational experiments

We devised a series of experiments to compare the performance of the heuristic procedures introduced in the previous section. The experiments consisted of a set of stochastic flow shop

problems in which the objective was to find the minimum expected makespan. To specify an instance of the problem, we had to specify the number of jobs and the (two) processing time distributions for each job. For simplicity, we assumed that all $2n$ processing time distributions were members of the same family, such as exponential, uniform, or lognormal. The exponential distribution is specified by a single parameter (its mean value) and corresponds to the case in which Talwar's Heuristic produces an optimal schedule. The uniform distribution is specified by two parameters (a mean value and a range) and allows us to influence the extent to which processing time distributions overlap. Finally, the lognormal distribution is the basis for most of the experimental work, for two reasons. First, the lognormal can also be specified by two parameters (a mean value and a standard deviation). Second, the lognormal has considerable practical value as a model for actual processing times. Among other traits, a lognormal variate is nonnegative, and its parameters can represent both high and low coefficients of variation.

The first step in generating a test problem was to obtain the parameters (the mean and, as needed, the standard deviation) for each of the $2n$ processing time distributions in the problem. These parameters were drawn randomly as samples from a pre-specified range of values. For example, we might obtain the mean processing times for eight jobs by sampling 16 times from the interval between 10 and 20. A similar drawing could provide the standard deviations for the processing times.

Once the parameters of the processing time distributions were determined, the next step was to construct the job sequence using one of the heuristic methods.

- To employ Talwar's Heuristic, the difference in processing rates was calculated for each job. Then a sequence was constructed in which these differences appeared in nonincreasing order.
- To employ Johnson's Heuristic, the mean values were treated as deterministic processing times. The desired sequence was then dictated by Johnson's Rule.
- To employ the API Heuristic, the values of the mean and standard deviation for each processing time were used in the pairwise comparisons of (5), and a stable sequence was derived. The process was initialized arbitrarily with the jobs in numerical order.

For comparison purposes, we also evaluated the sequence with jobs in numerical order. We might think of this sequence as "random" in that it uses no information about the processing time distributions in ordering the jobs.

The next step was to estimate the mean value of the makespan for each of the four sequences. This estimate was made based on a simulation containing 10,000 trials. Taking the minimum of these estimates, we identified the best sequence among the four that were considered.

This procedure was repeated (by sampling again for the parameters of the processing time distributions) ten times. For the ten replications, we recorded the average estimated mean makespan for each heuristic rule, the estimated standard deviation, the number of times (out of 10) that each heuristic produced the best estimated mean, and the average deviation between the heuristic procedure's estimated mean and the best value found.

In the remainder of the paper, we present several tables describing summaries of our various experiments. Here, we describe the notation in our tables.

- The top row of the table, labeled *Jobs*, shows the problem size and the family of distributions from which processing times were sampled.
- The second row, labeled *Means*, shows the range of values from which the various mean values were sampled.
- The third row, labeled *Ranges*, shows the range, in the case of uniform distributions, for the individual processing time distributions. This row is omitted in the exponential case because the mean value describes the variability in the distribution. In the case of the lognormal distribution, this row is labeled σ for the standard deviation of the distribution or *S.D.'s* for the range of standard deviations sampled.
- The remaining rows are labeled *R*, *J*, *T*, and *A*, corresponding to a Random sequence, Johnson's Heuristic, Talwar's Heuristic, and the API Heuristic, respectively. Across each row, the table contains four pairs of numbers, each pair giving our measure of average suboptimality and the number of times the procedure generated the best sequence.

To illustrate the type of data we compiled, we review one of our validation experiments, involving 6-job problems. In this experiment, we used the uniform distribution as a model for processing times. We drew mean processing times as integers from the interval (40, 60) and we took the range of the processing times to be 2 (or, equivalently, ± 1). Thus, except for the possibility that the means of two processing time distributions might match, the distributions did not overlap. Equivalently, during the simulation, if one processing time was longer than another on one simulated outcome, it had to be longer on all simulated outcomes. As a consequence, Johnson's Heuristic—applied to the mean processing times—should always produce an optimal sequence. We see that this is indeed the case in the first portion of Table 3 (for ± 1), which summarizes our results.

Table 3

Jobs	n = 6		Uniform					
Means	40-60		40-60		40-60		40-60	
Ranges	± 1		± 5		± 10		± 20	
R	3.01%	0	2.87%	0	3.34%	0	2.25%	0
J	0.00%	10	0.00%	9	0.03%	4	0.07%	4
T	0.14%	5	0.28%	5	0.17%	6	0.02%	8
A	0.23%	8	0.00%	10	0.01%	7	0.00%	8

In the table, we show the average suboptimality of the estimated mean makespan over ten replications, measured as the deviation from the best value among the heuristics tested. For the case in which the range of processing times was equal to the mean value ± 1 , Johnson's Heuristic (J) produced the best expected makespan each time (10 out of 10), as expected, so its average deviation was 0.00%. Talwar's Heuristic (T) produced the best value 5 times out of 10, and its average deviation was 0.14%. The API Heuristic (A) produced the best value 8 times out of 10, but its average deviation was higher at 0.23%. As a benchmark, the random sequence (R) never produced the best value and was on average about 3% worse than the best.

Having verified that our tests confirmed Johnson's Heuristic as the best when the processing time distributions did not overlap, we proceeded to increase the range of the processing time distributions, first to 10 and then to 20 and 40, repeating the experiment each time. The rest of Table 3 shows the results. For a range of 10, Johnson's Heuristic still produced an average deviation of 0.00%. As the range of the distribution was increased further, performance declined to 0.03% and 0.07%. This pattern suggests that Johnson's Heuristic is optimal when the processing time distributions do not overlap, but it is relatively less effective as overlap increases. Talwar's Heuristic exhibits somewhat the opposite behavior, with average deviations that drop as the overlap increases. For a range of 40, Talwar's Heuristic produced the best value 8 times out of 10.

Finally, the API Heuristic produced very good results in all cases. It produced the best value at least 7 times in each set of 10 replications, and its deviation was within 0.1% in all 40 test problems. It therefore demonstrated robust performance at different levels of variability.

To obtain a different perspective, we took an alternative parametric approach using a fixed range but sampling means from different intervals, as summarized in Table 4. With the range held fixed, the potential for overlap declines as we increase the interval from which mean values are drawn. Thus, we again see that Johnson's Heuristic fares best (with a deviation of 0.00%) when there is little or no overlap in the distributions, but its performance becomes worse than the other two heuristics when overlap is the greatest. Conversely, Talwar's Heuristic performs better when there is more overlap. Again, in this set of results, the API Heuristic demonstrates robust performance for all four experimental conditions. This general picture also emerged in experiments with more jobs.

Table 4

Jobs	n = 6		Uniform					
Means	45-55		40-60		20-80		10-90	
Ranges	± 20		± 20		± 20		± 20	
R	1.55%	0	2.11%	0	12.07%	0	14.66%	0
J	0.07%	3	0.01%	8	0.00%	9	0.00%	10
T	0.03%	8	0.28%	2	0.16%	7	0.27%	6
A	0.03%	6	0.07%	8	0.03%	8	0.00%	10

Table 4 also suggests that the performance of the Random sequence becomes relatively worse as the dispersion among the mean values increases. In the case of the interval (10, 90), the estimated deviation for the Random sequence was as large as 29% in one problem, which reminds us that in certain cases, intelligent heuristics can provide a substantial improvement over naïve scheduling in the stochastic flow shop.

In another validation experiment, we used the exponential distribution as a model for processing times. We drew mean processing times as integers from the ranges (10, 20) and in subsequent experiments, (10, 30), (10, 40), and (10, 50). In these cases, of course, all distributions overlap, a condition we expect to be unfavorable for Johnson's Heuristic. Moreover, from Theorem 2, Talwar's Heuristic is known to be optimal for the exponential case. Table 5 shows our results, which confirm the optimal performance. Johnson's Heuristic produces average deviations in the range from 0.1-0.2%, whereas the API Heuristic produces average deviations below 0.1%.

Table 5

Jobs	n = 6		Exponential					
Means	10-20		10-30		10-40		10-50	
R	3.17%	0	4.63%	0	5.62%	0	5.20%	0
J	0.20%	0	0.14%	4	0.16%	2	0.18%	3
T	0.00%	10	0.00%	10	0.00%	10	0.00%	10
A	0.03%	4	0.07%	5	0.09%	5	0.07%	5

For our main experiments, we used a lognormal distribution for processing times and worked with 8-job and 10-job problems. (We encountered no evidence that problem size was an important factor in the relative performance of the various heuristic methods, so we report only the results from the larger job sets.) Tables 6 and 7 summarize results for cases in which all processing time distributions have the same standard deviation. For relatively small standard deviations, such as $\sigma = 1$, Johnson's Heuristic performs best, probably because very little overlap exists. For $5 \leq \sigma \leq 20$, the performance of Johnson's Heuristic deteriorates rapidly, whereas Talwar's Heuristic improves. In Table 7, which shows results for $\sigma \geq 30$, we find that Talwar's Heuristic provides average deviations of 0.00%. The API Heuristic is nearly as good, but when the standard deviation reaches the mean of the distribution (which is also the case in exponential distributions), its performance is not as good as that of Talwar's Heuristic.

Table 6

Jobs	n = 10		Lognormal					
Means	40-60		40-60		40-60		40-60	
σ	1		5		10		20	
R	2.66%	0	2.20%	0	2.45%	0	2.06%	0
J	0.00%	10	0.02%	3	0.13%	2	0.22%	1
T	0.15%	1	0.18%	1	0.02%	4	0.00%	7
A	0.00%	10	0.00%	9	0.01%	6	0.00%	10

Table 7

Jobs	n = 10		Lognormal					
Means	40-60		40-60		40-60		40-60	
σ	30		40		50		100	
R	2.23%	0	2.65%	0	2.52%	0	2.09%	0
J	0.15%	1	0.23%	0	0.16%	0	0.14%	0
T	0.00%	9	0.00%	10	0.00%	10	0.00%	9
A	0.00%	9	0.00%	10	0.01%	4	0.01%	6

Table 8 displays the results for a set of experiments in which increasing levels of variation characterized both the dispersion of the means and the size of the standard deviations of the processing time distributions. This is perhaps our most general set of experimental conditions. In this series of outcomes, no one heuristic dominates, although Talwar's Heuristic appears to produce the best results overall. Table 9 shows a different perspective on the same experiments. Here, we show the worst-case deviation in each set of ten replications, along with

the estimated standard deviation of the makespan, averaged over the ten replications. Both measures represent ways of quantifying "risk" in making decisions under uncertain conditions. However, the worst-case deviations are all below 1% (except for the Random sequence), and the standard deviations are all quite close. Although the API heuristic tends to generate slightly higher standard deviations than Johnson's heuristic, the difference is small. Thus, the three heuristics do not appear to have substantially different properties when we consider decision-making under risk: the Johnson heuristic is slightly superior in terms of variance but the API heuristic is more robust in terms of the mean.

Table 8

Jobs	n = 10 Lognormal							
Means	40-60		30-70		20-80		10-90	
S.D.s	10-20		10-30		10-40		10-50	
R	3.06%	0	5.34%	0	6.71%	0	11.92%	0
J	0.15%	1	0.24%	1	0.18%	2	0.03%	6
T	0.01%	5	0.02%	7	0.00%	9	0.07%	6
A	0.03%	4	0.10%	2	0.14%	0	0.12%	0

Table 9

Jobs	n = 10 Lognormal							
Means	40-60		30-70		20-80		10-90	
S.D.s	10-20		10-30		10-40		10-50	
R	4.90%	47.46	9.70%	67.40	11.37%	87.24	20.67%	110.72
J	0.42%	47.02	0.80%	66.92	0.88%	84.81	0.18%	104.86
T	0.05%	46.60	0.08%	67.19	0.02%	84.59	0.53%	104.47
A	0.14%	47.19	0.42%	67.92	0.45%	85.73	0.32%	105.78

5 Summary and conclusions

We compared three heuristic methods for solving the stochastic two-machine flow shop problem. These three methods—Johnson's Heuristic, Talwar's Heuristic, and the API Heuristic—are structured around theoretical results in the analysis of flow shops, and each is relatively easy to implement in practice.

- We observed that Johnson's Heuristic is best when little or no overlap exists among the job processing time distributions. This might be the case if job processing times can be modeled by uniform distributions with modest size ranges or by normal distributions with small variances. When overlap exists, Johnson's Heuristic deteriorates compared to other heuristics, but its performance is consistently much better than that of an arbitrary sequence.
- We observed that Talwar's Heuristic is best when the distributions are all from the exponential family, a theoretical property that has already been noted in the literature. The more general form of this property, empirically speaking, is that the heuristic performs well when the processing time distributions exhibit substantial amounts of variation, and it is reliably better than an arbitrary sequence.
- We observed that the API Heuristic seems to have robust performance in many situations. It performs nearly as well as Johnson's Heuristic when little or no overlap exists, but it does not deteriorate as much when overlap is present. We observed no instance in which it produced a makespan as much as 0.5% above the best value found. However, it tends to yield makespan sequences with slightly higher variance than Johnson's heuristic.

Overall, the three heuristics require modest calculation effort and, when used in parallel, appear to generate solutions that are several percent better than those of an arbitrary solution procedure.

We might wonder about anchoring our quantitative results on the best solution found. One alternative might be to anchor the results on the solution to the deterministic counterpart,

which provides a valid lower bound [6]. However, the makespan of the deterministic counterpart is insensitive to variability and may thus be a misleading benchmark. A second alternative, of course, would be to anchor the results on the optimal value of the expected makespan. Because we have no known optimizing procedure for the stochastic flow shop, we would have to combine enumeration and simulation to produce an estimate, an approach that is computationally burdensome.

On an exploratory basis, we used a similar approach. We implemented a genetic algorithm as a solution technique, coupled with a simulation to evaluate sequences. We tested nine new 8-job instances, three each for the uniform, exponential, and lognormal distributions, and compared the solution found by Frontline Systems' Evolutionary Solver (ES) to the "best" solution found among our heuristics.² In all nine instances, the best heuristic solution was as good as or better than the ES solution, at least to within a 95% confidence interval as estimated by the simulation. We concluded that our best heuristic solution was either an optimum or very close to it.

We thus have three good heuristic tools for dealing with the two-job flow shop model. The next obvious step in developing scheduling procedures would be to create effective algorithms for three-machine and m -machine stochastic flow shop problems. Three types of approaches immediately come to mind:

- Based on the success of Talwar's Heuristic, we could create algorithms that rely on solving two-machine subproblems and use differences in processing rates, in the spirit of algorithms such as that of Campbell, Dudek, and Smith [8].
- Based on the success of Johnson's Heuristic, we could solve the deterministic counterpart of the n -job problem (for which several effective heuristics have been tested) and use the solution in the stochastic case.
- Based on the success of the API Heuristic, we could use neighborhood search schemes to identify potential improvements attainable from interchanging adjacent jobs in sequence.

With these tools, and the advent of fast simulation techniques, we should be able to generalize the current study and discover how to deal effectively with stochastic flow shops involving more than two machines.

References

1. Johnson, S.M., "Optimal Two- and Three-Stage Production Schedules with Setup Times Included," *Naval Research Logistics Quarterly*, 1, 61-68 (1954).
2. Ku, P.-S. and S.-C. Niu, "On Johnson's Two-Machine Flow Shop with Random Processing Times," *Operations Research*, 34, 130-136 (1986).
3. Makino, T., "On a Scheduling Problem," *Journal of the Operations Research Society Japan*, 8, 32-44 (1965).
4. Talwar, P.P., "A Note on Sequencing Problems with Uncertain Job Times," *Journal of the Operations Research Society Japan*, 9, 93-97 (1967).
5. Cunningham, A.A. and S.K. Dutta, "Scheduling Jobs with Exponentially Distributed Processing Times on Two Machines of a Flow Shop," *Naval Research Logistics Quarterly*, 16, 69-81(1973).
6. Portougal, V. and D. Trietsch, "Johnson's Problem with Stochastic Processing Times and Optimal Service Level," *European Journal of Operational Research*, 169, 751-760 (2006).
7. Baker, K.R. and D. Trietsch, *Principles of Sequencing and Scheduling*, Wiley (2009).

² The ES represents a highly sophisticated genetic algorithm that tends to be especially effective at solving sequencing problems, although it does not guarantee optimality.

8. Campbell, H.G., R.A. Dudek and M.L. Smith, "A Heuristic Algorithm for the n Job, m Machine Sequencing Problem," *Management Science*, 16, 630-637 (1970).

Optimal Pump Scheduling For Water Distribution Systems

Lindell Ormsbee • Srini Lingireddy • Don Chase

Abstract: Three different mathematical formulations for the scheduling of pumps to minimize energy consumption while still satisfying operational constraints are presented. Each formulation requires the solution of a highly nonlinear optimization problem that requires the iterative solution of an external numerical model for evaluating and satisfying the operational constraints. Two different algorithms for solving the problem are examined.

1 Introduction

Perhaps one of the most important, yet most widely overlooked components of the urban infrastructure in the United States, is the public water supply system. However, over the last several decades and in particular since 9/11, many utilities are beginning to move to the use of sophisticated computer technologies to not only provide critical real time information about the state of their systems, but also to improve their overall operations. One such area of focus has been in the optimal scheduling of pumps so as to minimize cost and improve operational conditions.

Researchers have been exploring the area of optimal pump operations for several decades [11]. In its most general form, the classical non-linear constrained optimization problem as applied to pump operations may be formulated as follows:

$$\text{Minimize or Maximize:} \quad F(X : q, p) \quad (1)$$

Lindell Ormsbee
University of Kentucky
Email: lormsbee@engr.uky.edu

Srini Lingireddy
Fluid Hammer Consultancy Service
Email: Srini@fluidhammer.com

Donald Chase
University of Dayton
Donald.Chase@notes.udayton.edu

$$\text{Subject to:} \quad q(X) = 0 \quad (2)$$

$$p(X) \geq 0 \quad (3)$$

$$X_{\min} < X < X_{\max} \quad (4)$$

where $F(X;q,p)$ represents the objective function (in terms of cost) to be minimized, $q(X)$ represents the explicit system constraints (conservation of mass and energy in terms of flow q), and $p(X)$ represents the implicit bound constraints (minimum pressures) to be satisfied. Finally, $X_{\min}$ and $X_{\max}$ represent explicit bound constraints on the decision variables of the optimal control formulation (duration of pump operations). The system constraints can be represented explicitly by the use of a simplified inductive model of the water distribution system [15] or a deductive model [10]. The exact form of the objective function and the associated decisions variables depend upon the nature of the problem formulation.

2 System Constraints

Because of the complexity of the system constraints, they are normally extracted out of the formal optimization formation and handled externally through use of a simulation program as shown in Figure 1. When using this structure, a vector of decision variables X is selected that explicitly satisfies the explicit bound constraints. This vector is then passed to the simulation program where the system constraints are satisfied and a vector of resulting pipe flowrates q and junction pressures p are determined. Once determined these values are passed back to the optimization program in order to check for feasibility of the implicit bound constraints (i.e. minimum pressures - p), and if feasible, the resulting value of the objective function associated with the decision vector X .

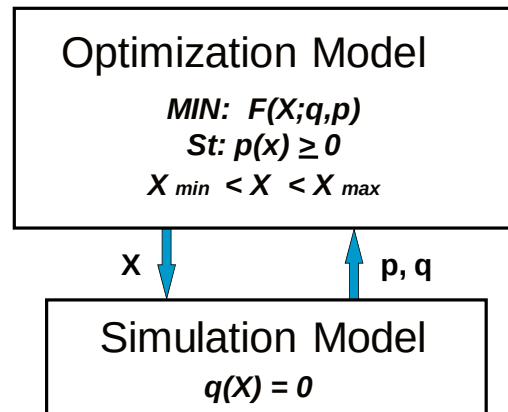
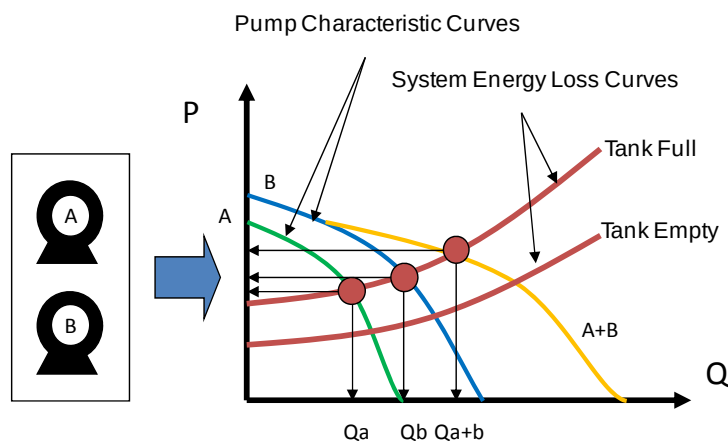


Figure 1. Problem Disaggregation Structure

The system constraints for this type of problem can be illustrated by considering the simple pipe distribution system shown in Figure 2. This system contains two tanks, five pipes, three junction nodes and one pump. The direction of flow in each of the pipes is indicated by the arrow on the line segment. In this case, a typical problem might be to determine the operating times of the pumps at the pump station over the course of an operating period (e.g. a day) so as to meet the varying demands M_i at junction node 3 while maintaining adequate operating pressures at each junction node (which is related to insuring that the water in the tank never drops below a critical level).

The majority of pumps used in the water industry are constant speed pumps, in which the discharge is a function of the water levels in the tank. During those times in which the pump station discharge exceeds M_t , the excess will fill the tank. During those times in which the pump station discharge is less than M_t , then the tank will drain. The decision of what pumps to turn on and how long they should run will be driven by both the nodal demand M_t and the tank level (which controls the associated junction pressures). In many cases, pump stations will consist of different size pumps with different flow versus pressure relationships. Thus, the decision of which pump or combination of pumps to run during a particular period can become rather complicated. This is further complicated by the fact that the discharges of multiple pumps cannot be simply added together. Instead, the combined flows are a nonlinear function of the energy loss through the system (as determined by a system energy curve) and the associated tank level. For example, consider the situation of two pumps as shown in Figure 3. Each pump will have a different discharge versus pressure curve (known as a pump characteristic curve) as shown in the accompanying graph. The actual discharge that a particular pump will produce is determined by the intersection point of the characteristic curve and the system energy loss curve. When pump A is operating, the discharge is Q_a . When pump B is operating the discharge is Q_b . However, when both pumps are operating, the discharge is $Q_a + b$. Technically, the system energy loss curve will shift up or down depending upon the water level in the tank, so things can become somewhat complicated.



343

The system energy loss curve represents a conceptual approximation of the flow versus pressure relationship that exists in a real water distribution network. In actual practice, this relationship must be determined by solving a series of nonlinear energy equations that are used to describe the physics of the system. For example, if we use the simple system shown in figure 2, then we are required to write one energy equation for each loop or energy path that exists in the network. An energy path is simply a path or series of pipes that connect any two tanks (technically any two points in the system with a known boundary condition – i.e. water level). For this simple system, the energy equation for the simple loop may be written as:

$$K_2 Q_2^{1.852} - K_4 Q_4^{1.852} - K_5 Q_5^{1.852} = 0 \quad (5)$$

Likewise the energy equation for the path may be written as:

$$K_1 Q_1^{1.852} - Z_p / Q_1 + K_2 Q_2^{1.852} + K_3 Q_3^{1.852} = \Delta E \quad (6)$$

where Q_i is the flowrate in pipe i , Z_p coefficient which is a function of the horsepower of the pump, ΔE is the difference in elevation between the water levels of the two tanks, and K_i is an energy loss term for each pipe that may be expressed as:

$$K_i = \frac{4.73 L_i}{C_i^{1.852} D_i^{4.87}} \quad (7)$$

where L = the pipe length in feet, D = the pipe diameter in feet, and C = an empirical roughness coefficient that depends on pipe age and pipe material (e.g. $C = 130$ for a new ductile iron pipe).

Since equations 5 and 6 are nonlinear in terms of Q , they cannot be solved directly but they may be solved iteratively using a Taylor's series approximation. For our simple system, the resulting equations become:

$$1.852[K_2 Q_2^{.852} - K_4 Q_4^{.852} - K_5 Q_5^{.852}] \Delta Q = K_2 Q_2^{1.852} - K_4 Q_4^{1.852} - K_5 Q_5^{1.852} \quad (8)$$

$$\begin{aligned} & [1.852 K_1 Q_1^{.852} + Z / Q_1^2 + 1.852 K_2 Q_2^{.852} + 1.852 K_3 Q_3^{.852}] \Delta Q = \\ & = K_1 Q_1^{1.852} - Z p / Q_1 + K_2 Q_2^{1.852} + K_3 Q_3^{1.852} + \Delta E \end{aligned} \quad (9)$$

If we express equations 8 and 9 in matrix notation, then the set of equations can be written as:

$$\begin{bmatrix} G_I(Q_I) & 0 \\ 0 & G_{II}(Q_{II}) \end{bmatrix} \begin{bmatrix} \Delta Q_I \\ \Delta Q_{II} \end{bmatrix} = \begin{bmatrix} -F_I(Q_I) \\ -F_{II}(Q_{II}) \end{bmatrix}$$

Figure 4. Matrix structure for simple network example

where G_I and F_I represents the function on the right hand side and left hand side of equation 8 respectively, and Q_I represents the vector of flows. Likewise, G_{II} and F_{II} represent the functions on the right hand side and left hand side of equation 9 respectively, and Q_{II} represents the vector of flows. Reducing further, the equations can now be expressed as:

$$[G]\Delta Q = \{F\} \quad (10)$$

Thus, for this simple example, each time the simulation model is accessed by the optimization model, initial estimates of the flows in each pipe must be generated so that conservation of mass is preserved at each junction node (e.g. $Q_1 - Q_2 - Q_4 = 0$). Once this is done, the K coefficients are determined and substituted into the equations. Next, flow adjustment factors (i.e. ΔQ) are calculated for each loop or path as follows:

$$\Delta Q = [G]^{-1} \{F\} \quad (11)$$

Once these are determined, each of the individual flows in each pipe may be determined using a recursive equation. For example, for pipe 1 the flows are updated as follows:

$$Q_{1i+1} = Q_{1i} + \Delta Q_I \quad (12)$$

For pipes that are common to more than one equation (e.g. pipe 2), the recursive equation is expressed as follows:

$$Q_{2i+1} = Q_{2i} + \Delta Q_I + \Delta Q_{II} \quad (13)$$

Once all the flows have been updated, they are then used to update the K coefficients and the process is repeated until the ΔQ 's converge to zero.

It should be recognized that most water distribution systems are much more complicated than the simple network shown in Figure 2. Further, in simulating the performance of such a system over time (e.g. a day), the previous solution methodology must be repeated for each time step in the simulation. In this case, the flows determined at the beginning of each time step are then used to update the boundary conditions for use in solving for the flows at the next time step. In order to minimize the error associated with this type of Euler integration, a maximum time step of one hour may be necessary. When applied to typical water distribution system, the computational requirements associated with each "function call" from the optimization model can become significant.

3 Objective Function

The objective of the extended period operation problem is to minimize the total energy consumption charges associated with operating a set of pumps over the course of an operating horizon, while simultaneously satisfying any service and reliability-related requirements of the system. In a typical water distribution system, the energy consumption costs incurred by pumping water depends mainly on the rate at which water is pumped, the associated pump head, the duration of pumping, the unit cost of electricity, and the different combined efficiencies of various pump combinations. Mathematically the objective function may be expressed as:

$$\text{Minimize } Z = \left(\frac{0.746\gamma}{550} \right) \sum_{t=1}^T R_t \sum_{i=1}^I \left[\frac{Q_{t,i} H_{t,i} X_{t,i}}{e_{t,i}} \right] \quad (14)$$

where: Z = the total energy cost to be minimized (\$)

$Q_{t,i}$ = the average flowrate associated with pump i during time t (cfs)

$H_{t,i}$ = the average head associated with pump i during time t (ft)

note: the term “head” is a common term in pump hydraulics used to express pressure in terms of an equivalent height of water.

$X_{t,i}$ = the duration of the time pump i is operating during interval t (hr)

$e_{t,i}$ = the average wire to water efficiency associated with pump i during time t

R_t = the electric rate during time t (\$/Kw-hr)

γ = the specific weight of water (lb/ft³)

I = the total number of pumps included in the optimization

T = the total number of time intervals in the operating horizon

For a given network configuration and an associated set of initial boundary conditions (including the vector of initial tank levels E and the vector of system demand loadings M), the average discharge Q_i , pump head H_i , and pump efficiency e_i , associated with a particular pump i can be expressed as a function of the characteristics of the pump itself plus the characteristics of another pump which may be operating during the same time periods as pump i . Since the set of pumps operating during a particular period, t , can be explicitly defined by the duration of time each pump in the set is operating, (i.e. if $X_{t,i} = 0$ then pump i is not operating during time period t and if $X_{t,i} > 0$ then the pump is operating during period t), then the pump discharge, the pump head, and the pump efficiency can be expressed as implicit functions of the vector of total pump durations for a particular time interval [12]. As a result, equation 14 may now be expressed as:

$$\text{Min } Z = \sum_{t=1}^T \sum_{i=1}^I f \left[Q_{t,i}(\mathbf{X}_t, \mathbf{M}_t, \mathbf{E}), H_{t,i}(\mathbf{X}_{t,S}, \mathbf{M}_t, \mathbf{E}_t), e_{t,i}(\mathbf{X}_{t,S}, \mathbf{M}_{t,i}, \mathbf{E}_{t,i}), \mathbf{X}_{t,S}, \mathbf{R}_t \right] \quad (15)$$

As a result, the objective function can be expressed solely in terms of the vector of the individual pump operating times. As will be discussed in the following sections, the exact nature of the pump operating times will be dependent upon the problem formulation.

2 Implicit Scheduling Formulation

Historically, the optimal pump scheduling problem has been either formulated as an implicit control problem or an explicit control problem. In the implicit formulation, pump station discharge, supply pressure, or tank water levels are treated as the decision variables of the optimal control problem. In the explicit formulation, the actual pump operating times (either individually or compositely) are treated as the decision variables.

The implicit formulation will typically require the solution of two sub-problems. The first sub-problem involves determining an optimal decision trajectory. The optimal decision trajectory can be defined as that series of pump station discharges, supply pressures, or tank water levels which, over the course of the operating horizon, result in a minimal total operating cost.

The second sub-problem involves determining the specific pump operations which produce the optimal decision trajectory. The difficulty associated with finding specific pump combinations is compounded by the fact that many combinations capable of producing the desired decision trajectory may exist. Furthermore, from all the possible combinations capable of producing the desired trajectory, the combination which results in minimal operating costs must be found.

Historically, the implicit approach has been used to develop control strategies for single tank systems. In such cases, the tank water level has been used as the implicit decision variable and the resulting formulation is typically solved using dynamic programming [7], [8], [13], [16]. Due to the curse of dimensionality, dynamic programming solutions are normally restricted to problems involving no more than three storage tanks. For systems with several tanks, researchers have either employed dynamic programming along with an associated site specific spatial decomposition scheme [3], [21] or nonlinear programming along with an alternative implicit decision variable [9].

3 Discrete Explicit Pump Scheduling

In the discrete explicit approach, the actual times of the operation of each pump are treated as the decision variables [6], [14]. In this case, the objective function may be characterized in terms of operational costs and the associated state variables (such as flow or pressure) can then be modeled using a much more robust model of the water distribution system that can be linked to the optimization algorithm via iterative subroutine calls [19].

3.1 Restricted Formulation.

The restricted formulation of time as a decision variable was originally proposed by Chase and Ormsbee [5] and later applied by Chase [4] and Brion [2]. In this approach, a pump is forced to begin operating at the beginning of a pre-determined timer interval (e.g. every four hours), in effect, restricting the time a pump can begin operating. The decision variable in this case is the duration of time each pump operates during a particular time interval. For example, suppose decision variable $X_{3,2}$ has a value of 2.50 and a four hour time interval is used. Under the restricted approach, pump 2 would be turned on at the beginning of time interval #3 and turned off 2.5 hours into the interval. In other words if the operating horizon begins at midnight, pump 2 would be placed on line at 8:00 a.m. and turned off at 10:30 a.m. The values taken on by the decision variables are bounded between 0 (pump off) and Δt (duration of time interval). Figure 5 provides a pictorial representation of the decision variables in the restricted approach.

As can be seen from Figure 5, the total number of decision variables required to solve the optimal control problem is equal to the product of the number of pumps and the number of time intervals making up the operating horizon. As the size of the time interval is decreased, the closer the problem becomes to one of continuous pump operation. Unfortunately a decrease in the size of the time interval results in a proportional increase in the number of time intervals which, in turn, causes an increase in the number of decision variables. An increase in

the number of decision variables can be undesirable since the computation time required to solve the optimal control problem will increase.

3.2 Unrestricted Formulation.

In an effort to reduce the number of decision variables, yet continue to pose the problem within the framework of an explicit time model, the restricted formulation can be modified to yield an unrestricted formulation. In the unrestricted formulation, the decision variables become the starting and ending time for the pumps. Figure 6 shows a pictorial representation of the decision variables for the unrestricted formulation.

Unlike the restricted formulation, in the unrestricted formulation there are fewer conditions as to when a pump may start or stop. Instead of forcing a pump to begin operating at the start of a particular time interval, pumps are allowed to start operating at any time during the operating horizon. Likewise, instead of shutting down a pump during a given time interval perhaps only to have it operating again at the start of the next time step, under the unrestricted approach a pump may stop operating at any time during the operating horizon. Consequently, feasible values of the decision variables for the unrestricted approach are bounded between 0 (pump off) and T where T is the operating horizon, usually 24 hours.

3.3 Discussion of Unrestricted and Restricted Formulations

A nice feature of the unrestricted approach is that it more closely replicates actual pump operation. In other words, system operators will typically place pumps on line in response to certain system parameters such as a low tank level, low pressure, an increase in system demands, a fire, etc. Such changes in system indices and/or system loadings are not likely to occur at the start of a pre-defined time interval, unless of course, the time interval is extremely small.

The restricted formulation allows pumps to operate several times per day. This can be accommodated in the unrestricted formulation by increasing the number of decision variables. In this case, a pair of decision variables are associated with each pump duty cycle. If a pump is allowed to operate only once per day, then there is one pump duty cycle per day and the number of decision variables is twice the number of pumps, i.e. one decision variable for the starting time and one for the ending time. If all pumps are allowed to operate twice per day, then there are two pump duty cycles for each pump. Following the convention that a pair of decision variables is associated with each pump duty cycle, then the total number of decision variables would be four times the number of pumps.

In general, the number of decision variables associated with the unrestricted formulation can be found using the following equation:

$$NDV = \sum_{i=1}^I (2 * DC_i) \quad (16)$$

where NDV is the total number of decision variables in the unrestricted explicit formulation, DC_i is the number of daily duty cycles allowed for pump i, and I is the number of pumps included in the optimization.

With the unrestricted formulation, additional bound constraints are needed to assure that the starting time for a pump is less than the stopping time for the same pump. If multiple duty cycles are allowed, then constraints are needed to insure that the starting time for particular

pump duty cycle is greater than the stopping time for the pump's previous duty cycle. The additional constraints required for the unrestricted formulation can be expressed with the following equations:

$$X_{i+I} - X_i > 0 \quad i = 1, 2, \dots, I \quad (17)$$

$$X_{n,I} - X_{n-1,i+I} < 0 \quad i = 1, 2, \dots, I; n = 2, 3, \dots, DC \quad (18)$$

where X_i is the starting time for pump i , X_{i+1} is the stopping time for pump i , $X_{n,i}$ is the starting time for pump i for the n th duty cycle, and $X_{n,i+1}$ is the stopping time for pump i during the n th duty cycle. Such constraints are not necessary for the restricted approach since the decision variable is the duration of time a pump operates and since the formulation keeps pump operation within a single time interval.

Pump 1						
Pump 2						
Pump 3						
Pump 4						
Pump 5						
Pump 6						
Time	0 – 4 hrs	4 – 8 hrs	8 – 12 hrs	12 – 16 hrs	16 – 18 hrs	18 – 24 hrs

Figure 5 Schematic Representation of Decision Variables for the Restricted Approach

Pump 1						
Pump 2						
Pump 3						
Pump 4						
Pump 5						
Pump 6						
Time	0 – 4 hrs	4 – 8 hrs	8 – 12 hrs	12 – 16 hrs	16 – 18 hrs	18 – 24 hrs

Figure 6. Schematic Representation of Decision Variables for the Unrestricted Approach

Two separate explicit control formulations have been proposed: 1) a restricted formulation and 2) an unrestricted formulation. Under the restricted formulation, pumps are forced to begin operating at the start of a time step and the value of the decision variable is equal to the duration of time the pump operates. Under the unrestricted formulation, pumps are allowed to start and stop operating at any time within the operating horizon and the decision variables represent the actual starting and stopping times for each pump duty cycle. With respect to the restricted formulation, the unrestricted approach allows more freedom of operation and more closely resembles actual pump operation. However, the unrestricted approach requires the use of additional constraints to regulate pump operation within practical limits.

As a result, the addition of new constraints results in a slight modification of the optimal control problem originally shown in equations 1-4. The modified optimal control problem for the explicit formulation may now be expressed as:

$$\text{Minimize or Maximize:} \quad F(X) \quad (19)$$

$$\text{Subject to:} \quad G_j(X) \leq 0 \quad j=1, 2, 3 \dots J \quad (20)$$

$$H_k(X) = 0 \quad k=1, 2, 3 \dots K \quad (21)$$

$$U_m(X) \leq 0 \quad m=1, 2, 3 \dots M \quad (22)$$

$$X_{\min} \leq X \leq X_{\max} \quad (23)$$

where all the terms are the same as before and $U(X)$ represent bound constraints used to keep pump starting and stopping times within practical limits when using the unrestricted approach. If constraints (20) – (23) are included directly in the optimization problem as binding constraints, then the result for most real problems would be a large scale nonlinear problem. The scale of the problem can be reduced by handling the constraints in the following manner. The implicit system constraints, $H(X)$, are solved through the use of a simulation model [20]. Depending upon the size of the mathematical model describing the water distribution systems, accommodating the implicit constraints through the use of a simulation model can substantially reduce the number of constraints which would otherwise be solved by the optimization algorithm. The implicit bound constraints, $G(X)$, and the bound constraints associated with the unrestricted approach, $U(X)$, can be included in the objective function as a penalty term so as to allow solution using an unconstrained optimization method. Alternatively, depending upon the type of solution algorithm employed, they may also be explicitly satisfied outside of the algorithm. Finally, the explicit bound constraints can be used to assign values of decision variables when the variables exceed their bounds. For example, if during the course of the optimization a value taken on by a decision variable is greater than the upper bound of the variable, $X_{\max}$, then the value of the decision variable is set equal to the upper bound. Similarly if a value of a decision variable is less than the lower bound, $X_{\min}$, then the value of that variable is set equal to its lower bound.

4 Composite Explicit Pump Scheduling

As discussed previously, the exact form of the decision variable will be dependent upon whether a restricted or an unrestricted formulation is employed. In applying the restricted

approach, the normal operating horizon (typically 24 hours) is divided into T separate time intervals (e.g. 4 hours) and the pump operating time for each pump in each time interval is determined. In the unrestricted approach, a specific number of pump duty cycles is specified for each pump and the beginning and ending times of each duty cycle is determined. For distribution systems with multiple pump stations, and with each pump station containing numerous pumps, both formulations can result in an excessive number of decision variables. One way to significantly reduce the total number of decision variables would be to develop a single decision variable for each pump station for each time interval that relates the particular set of pumps in operation during that period. Such a formulation can be obtained by ordering the various available pump combinations associated with each pump station on the basis of unit cost. A single decision variable can then be developed for each pump station s and each time interval t of the form $X_{st} = II.CC$ where II is an integer and corresponds to the identification number of the pump combination that operates $CC\%$ percent of the time interval. By definition, it will also be assumed that combination $II-1$ operates the remaining $(1-CC)$ percent of the time interval. (The combination in which $II = 0$ corresponds to the null combination or the decision to run no pumps). Modification of the original objective function to accommodate the proposed formulation yields the following objective function:

$$\text{Min } Z = \sum_{t=1}^T \sum_{i \in S} f \left[Q_{t,i}(X_{t,s}, \mathbf{M}_t, \mathbf{E}_t), H_{t,i}(X_{t,s}, \mathbf{M}_t, \mathbf{E}_t), e_{t,i}(X_{t,s}, \mathbf{M}_{t,i}, \mathbf{E}_{t,i}), X_{t,s}, \mathbf{R}_t \right] \quad (24)$$

The objective function as expressed in Eq. 24 is subject to the same three sets of constraints as with the previous explicit formulations. These include: (1) a set of implicit system constraints, (2) a set of implicit bound constraints, and (3) a set of explicit decision variable constraints. While both the implicit system and bound constraints will have an identical form as before, the explicit decision variable constraints are different for the new formulation as a result of the use of a new set of decision variables. In this case the decision variable for each pump station for a particular time interval will be restricted between a lower value of zero (corresponding to no pumps in operation) and an upper value related to the maximum number of pump combinations available for that pump station.

In applying the proposed algorithm to a specific distribution system, the desired operating horizon (typically 24 hours) is once again divided into a discrete set of time intervals. A separate decision variable for each time interval is then assigned to each pump station. To initiate the algorithm a separate vector for each time interval is randomly generated or explicitly specified which contains the values of the decision variables for each pump station in the system. As a result, any potential solution will consist of a set of N vectors where N – the number of time intervals which constitute the operating horizon. To insure a feasible solution, the initially specified or generated set of decision vectors must satisfy the explicit bound constraints.

Similar to the previous explicit control formulations, the proposed composite formulation also uses a disaggregated solution methodology. That is, once an initial set of decision vectors is obtained, it is then passed down to a network simulation model [20] for use in explicitly satisfying the implicit system constraints and for use in evaluating the implicit bound constraints. The values of the resulting state variables (i.e. flowrate, pressure, kilowatt consumption, etc.) are then passed back to the optimization algorithm for use in quantifying the objective function and identifying any violations in the implicit bound constraints. This information is then used to generate an improved set of decision vectors which automatically satisfies the explicit bound constraints and which seeks to minimize the objective function.

Once generated, the improved set of decision vectors is then passed back down to the simulation algorithm for subsequent evaluation. This process is then repeated until a specified level of algorithmic convergence is obtained.

To permit increased model flexibility, the computational time interval used in the simulation model is not restricted to be equal to the interval associated with the decision variables. For example, if a 12 hour time interval is used for the decision variables a much smaller time step (i.e. 2 hour, 4 hour, etc.) can be used as the time interval in the simulation model.

5. Solution Methodologies

Various optimization methods have been employed by researchers in solving the discrete explicit pump scheduling problem. These methods have varied from traditional gradient based methods to more exotic evolutionary methods (e.g. genetic algorithms). More recently Tufail [18] and Tufail and Ormsbee [19] have proposed using a modified and improved version of a relatively straightforward direct search method [1]. The approach, called the Shuffled Box Complex Method, has a potential advantage over genetic algorithms in that both implicit and explicit constraints may be handled directly without the use of a penalty formulation. Similar to genetic algorithms, their method also pursues an optimal solution along multiple simultaneous search paths, thereby improving the efficiency of the original method.

While the composite explicit pump scheduling problem can also be solved using the same algorithms discussed above, Ormsbee and Reddy [12] found that the unique nature of the problem yields itself to solution via a simple heuristic. Because of the nature of the selected decision variable, the least cost solution to the unconstrained objective function (i.e. Eq. 24) is explicitly known. That is, the least cost solution is one in which no pumps are operated. As a result, the optimization problem reduces to one of finding the set of decision vectors which produce solutions on the composite constraint boundaries that are as close to the origin of the n dimensional solution space as possible. For a given initial set of decision vectors an improved set may be obtained by simply contracting the scalar values in the vectors toward the origin. In the event that a contraction results in a constraint violation, then the vector that produced the constraint violation can be subsequently expanded until a feasible solution is obtained. By continuing to bisect the resulting search vector, a set of decision variables can be obtained which will result in a solution which lies just on the constraint boundary. In the event the initial solution violates an implicit bound constraint, then the solution may be expanded away from the origin along the resulting search direction until a feasible solution is obtained. Due to the nature of the explicit bound constraints, a feasible solution will eventually be found.

It should be recognized that application of such an approach will only result in the best solution that lies on the search direction located between the given initial solution and the origin of the dimensional solution space. However, additional feasible solutions may be obtained by simply replicating the methodology using additional sets of initial decision vectors. Once the final set of feasible solutions is obtained an "optimal" solution may be obtained by simply selecting the best solution from among the resulting feasible solutions.

More recently, Teegavarapu et al. [17] combined the general search heuristic with a genetic algorithm (GA) by using the GA to continue to generate a new set of potentially feasible solutions. Once again, the algorithm is restricted to that class of problems where the unconstrained optimal solution is known (e.g. where all the pump operating times are zero). In this case, once the new population of solutions is generated, each solution is checked for feasibility. Those solutions that are not feasible are projected up to the constraint boundary

from below while those solutions that are feasible are now projected down to the constraint boundary from above (Figure 7). This will then insure that each offspring in the current population is both feasible and locally optimal (e.g. relative to the vector that originates from the origin of the solution space). Once these solutions are generated, the GA is now engaged and a new population is generated (Figure 8). Once again, some of these solutions will be feasible and some will be infeasible. As before, both sets of solutions are projected to the constraint boundary, and the process is continued (Figure 9). Eventually, the GA will drive the solutions to that region on the constraint boundary that is closest to the origin of the solution space and thereby yield an optimal solution. (Figure 10).

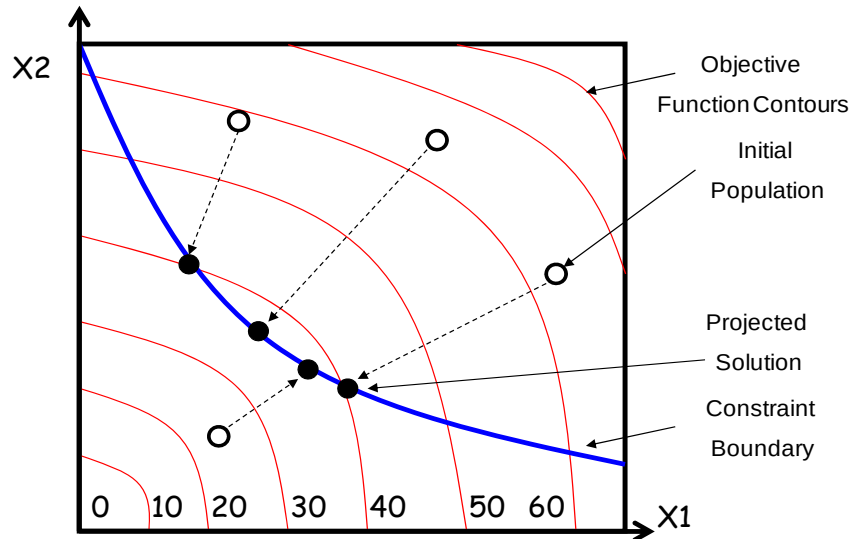


Figure 7. Projection of Initial Solutions to Constraint Boundary (2 dimensional example)

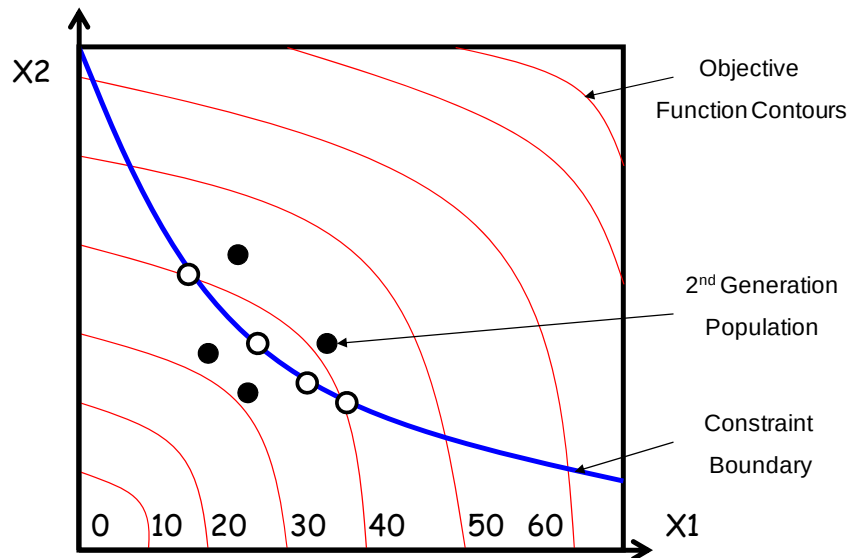


Figure 8. Generation of New Population (2 dimensional example)

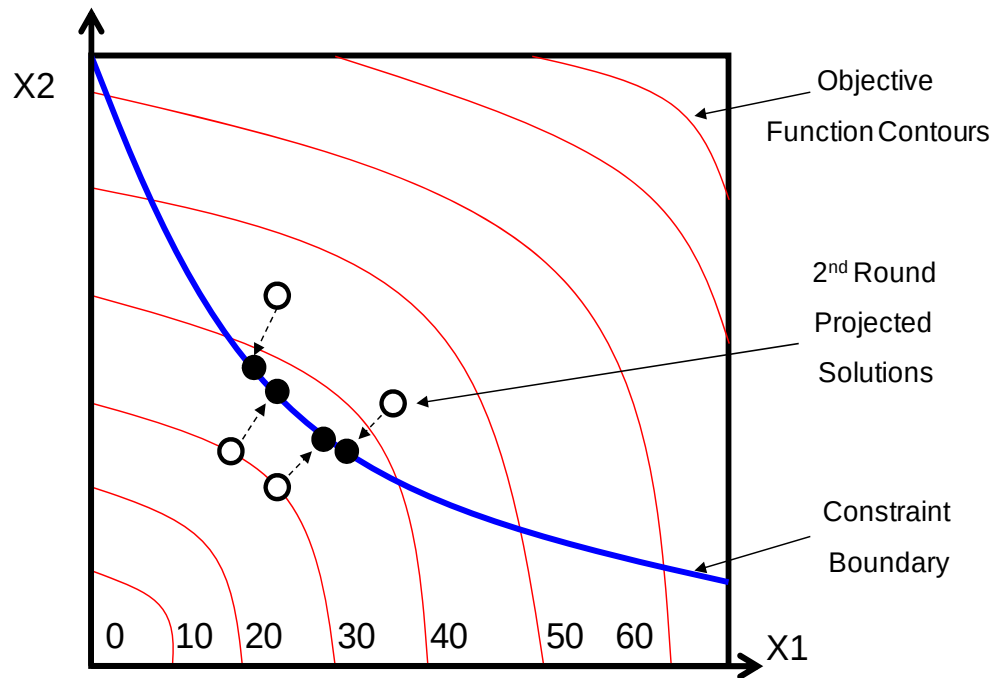


Figure 9. Projection of 2nd Round Solutions to Constraint Boundary (2 dimensional example)

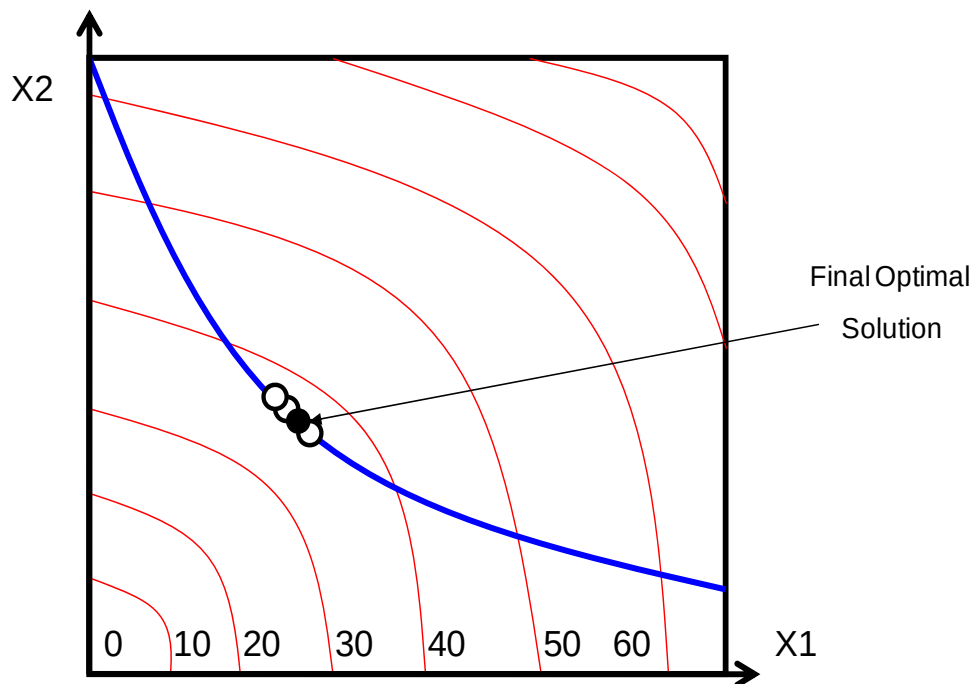


Figure 10. Collapse of Nth Round Solutions to Optimal Solution (2 dimensional example)

6 Summary

Three different explicit formulations of the optimal pump scheduling problem have been presented. The resulting formulations may be solved by using either unconstrained methods along with penalty terms or constrained methods that explicitly incorporate the constraints via the mechanisms involved in the algorithm. In either case, the implicit system constraints can be solved directly using an external simulation program which is linked to the optimization algorithm via subroutine calls. Given the complexity of the system being modeled (i.e. water distribution systems), it is felt that the provided formulations as well as the explicitly constrained GA may have applicability to other complex scheduling problems as well.

7 References

1. Box, M. J. A new method for constrained optimization and a comparison with other methods. *Computer Journal*, Vol. 8, No. 1, 42-52 (1965).
2. Brion, L., M., *Methodology for Optimal Pumping Stations in Water Distribution Systems*, Ph.D., Dissertation, University of Texas at Austin (1990).
3. Carpentier, P., and G. Cohen, "Decomposition, Coordination and Aggregation in the Optimal Control of a Large Water Supply Network," *Proc. of the 9th Triennial IFAC World Congress*, Budapest, pp. 3207-3212, (1984).
4. Chase, D., V., "A Computer Program for Optimal Control of Water Supply Pump Stations: Developing and Testing," USACERL Technical Report N-90/14, U.S. Army Corps of Engineers, Construction Engineering Research Laboratory, Champaign, Illinois, pp. 1-99 (1990).
5. Chase, D., and L. Ormsbee, "Optimal Pump Operation of Water Distribution Systems with Multiple Storage Tanks," *Proceedings of the Amer. Water Works Association. Computer Specialty Conf.*, Denver, CO, pp. 205-214 (1989).
6. Chase, D., and Ormsbee, L., "An Alternate Formulation of Time as a Decision Variable To Facilitate Real-Time Operation of Water Supply Systems", *Proceedings of the 18th Annual Conference of the ASCE Water Resources Planning and Management Division*, New Orleans, LA, pp. 923-927 (1991).
7. Coulbeck, B., "Optimisation of Water Networks," *Transactions of the Institute of Measurement and Control*, Vol. 6, No. 5, pp. 271-279 (1984).
8. DeMoyer, R., and L. Horowitz, *A Systems Approach to Water Distribution Modeling and Control*, Lexington Books, pp. 143 (1975).
9. Lansey, K., and Q. Zhong, "A Methodology for Optimal Control of Pump Stations," *Proceedings of the ASCE Water Resources Planning and Management Specialty Conference*, Fort Worth, TX, pp. 58-61 (1990).
10. Ormsbee, L.E. (1985). "OPNET: A Nonlinear Algorithm for Hydraulic Networks", *Proceedings of the ASCE Water Resources Planning and Management Conference: Computer Applications in Water Resources*, Buffalo, New York, pp. 739-748 (1985).

11. Ormsbee, L.E., and Lansey, K.E., "Optimal Operation of Water Distribution Systems, *ASCE Journal of Water Resources Planning and Management*, Vol. 120, No. 2, pp. 237-252 (1994).
12. Ormsbee, L.E., and Lingireddy, S. L., "Nonlinear Heuristic for Pump Operations," *ASCE Journal of Water Resources Planning and Management*, Vol. 121, No 4, pp. 302-309 (1995).
13. Ormsbee, L.E., Walski, T., Chase, D., Sharp, W., "Methodology for Improving Pump Operation Efficiency," *ASCE Journal of Water Resources Planning and Management*, Vol., 115, No. 2, pp. 148-164 (1989).
14. Ormsbee, L., Reddy, S., and Chase, D., "Comparison of Three Nonlinear Control Algorithms for the Optimal Operation of Water Supply Pumping Systems, *Proceedings of the International Conference on Integrated Computer Applications for Water Supply and Distribution*, Leicester, England, pp. 259-271 (1993).
15. Ormsbee, L., and Reddy, S., "Pumping System Control Using Genetic Optimization and Neural Networks," *Proceedings of 7th IFAC/IFORS/IMACS Symposium on Large Scale Systems: Theory and Applications*, London, England, (1995).
16. Sabet, M., and O. Helweg, "Cost Effective Operation of Urban Water Supply System Using Dynamic Programming," *Water Resources Bulletin*, AWRA, Vol. 21, No. 1, pp. 75-81 (1985).
17. Teegavarapu, R., Bradley, S., and Ormsbee, L., "Probabilistic Goal Driven Watershed Management", *Proceeding of the World Environmental and Water Resources Congress*, Tampa, May 15-19, (2007).
18. Tufail, M., *Optimal Water Quality Management Strategies for Urban Watersheds Using Macro-Level Simulation Models Linked with Evolutionary Algorithms*, Submitted in Partial Fulfillment of the Doctor of Philosophy Degree, University of Kentucky, Lexington, Kentucky, 40506 (2006).
19. Tufail, M., and Ormsbee, L.E., "Optimal Water Quality Management Strategies for Urban Watersheds Using Macro-Level Simulation and Optimization Models, *ASCE Journal of Water Resources Management*, Vol. 135, No. 4, pp. 276-285, (2009).
20. Wood, D. J., User's Manual – Computer Analysis of Flow in Pipe Networks Including Extended Period Simulations, Department of Civil Engineering, University of Kentucky, Lexington, KY, (1980)
21. Zessler, U., and U. Shamir, "Optimal Operation of Water Distribution Systems," *ASCE Journal of Water Resources Planning and Management*, Vol. 115, No. 6, pp. 735-752 (1989).

Single Machine Scheduling with Interfering Job Sets

Ketan Khowala, John Fowler, Ahmet Keha, Hari Balasubramanian

Abstract We consider a bicriteria scheduling problem for a single machine in which jobs belong to either of two different disjoint sets, each set having its own performance measure. The problem has been referred to as interfering job sets in the scheduling literature and also been called multi-agent scheduling where each agent's objective function is to be minimized. The performance criteria specifically selected in this paper are minimizing total completion time and number of tardy jobs for the two set of jobs. We present a forward SPT-EDD heuristic to generate the set of all non-dominated solutions for this problem considering a single machine environment where all the jobs are available at time zero and the jobs are not allowed to be preempted. The complexity of this specific problem is still open; however some pseudo-polynomial algorithms have been suggested by earlier researchers which have been used to compare the results from the proposed heuristic.

1 Introduction

Motivated by multiple objectives and tradeoffs that a decision maker has to make between conflicting objectives, multicriteria scheduling problems have been widely dealt with in the literature. In this domain of scheduling problem one typically has to satisfy multiple criteria on the same set of jobs. However, in some cases jobs belong to different job sets and must be processed using the same resource (or competing for the same machine), hence causing interference. These job sets can have different criteria to be minimized. These different sets may represent different customers or different agents whose requirements may differ. The complexity of this domain of problems can be attributed to the number of different job sets considered, performance criteria and restrictions on each set of jobs and the machine environment.

One of the earliest references on this subject is Peha [1995] which dealt with the problem of interfering job sets with objectives of minimizing weighted number of tardy jobs in one set and total weighted completion time in another set of jobs with unit processing time under an identical parallel machine environment. The assumption of unit processing times makes the problem easier to solve. The paper from Baker and Smith [2003] was the first paper formalizing scheduling problems with interfering job sets. They considered a single machine

Ketan Khowala
Arizona State University
E-mail: Ketan.Khowala@asu.edu

John Fowler
Arizona State University
E-mail: John.Fowler@asu.edu

Ahmet Keha
Arizona State University
E-mail: Ahmet.Keha@asu.edu

Hari Balasubramanian
University of Massachusetts
E-mail: hbalasubraman@ecs.umass.edu

problem involving the minimization of criteria including makespan, maximum lateness, and total weighted completion time ($C_{\max}, L_{\max}, \sum w_j C_j$). They showed that any combination of these criteria on different job sets can be solved in polynomial time by defining the optimization function as a linear combination of the criteria on different job sets, except the combination of $\sum w_j C_j$ and $L_{\max}$ on different job sets which turns out to be NP-hard.

Agnetis et al. [2004] presented the complexity of generating non dominated solutions for single machine as well as shop floor scheduling problems with interfering job sets, involving objectives of minimizing total completion times, total number of tardy jobs and total weighted completion times ($\sum C_j, \sum U_j, \sum w_j C_j$). Ng et al. [2006] proved that the problem involving the jobs sets with total completion time and total number of tardy jobs on a single machine is NP-hard under high multiplicity encoding and have presented a pseudo polynomial time algorithm for this problem. Cheng et al. [2006] have shown NP completeness of the problem where jobs belong to one of the multiple sets and each set is having the objective of minimizing the total weighted number of tardy jobs ($\sum w_j U_j$). They have also presented a polynomial time approximation scheme for this problem. In one of the most recent works, Balasubramanian et al. [2008] presented a heuristic to generate all the non dominated points for the interfering job sets on parallel machine environment. The paper proposes an iterative SPT-LPT-SPT heuristic and a bicriteria genetic algorithm for the problem by exploiting the structure of the problem and a comparison of the computational efficiency including the time index MIP formulation as well.

In the subsequent sections, we define our problem, talk about the structure of the problem and some key properties of the problem, present the forward SPT-EDD heuristic, and compare the computational performance of the non dominated solution sets obtained from our heuristic with the Pareto optimal solution sets obtained by the pseudo polynomial algorithm by Ng et al. [2006].

2 Problem Description

The specific problem that we are looking into relates to single machine scheduling where all jobs are available at time zero and no preemption is allowed. We look at the single machine problem with interfering jobs from two disjoint sets, one having the objective of minimizing total completion time and other minimizing total number of tardy jobs. As discussed earlier, the complexity of this problem is still open. A pseudo- polynomial algorithm is presented by Ng et al. [2006] for this problem under binary encoding. We combine some of the intuition from Moore's algorithm to determine the initial set of jobs that can be on time and then use a forward SPT-EDD heuristic to determine all the non dominated points for this problem.

There are two disjoint interfering sets of jobs ξ_1 and ξ_2 with n_1 and n_2 number of jobs in each respective set. The total number of jobs that need to be scheduled is $n = (n_1 + n_2)$. We seek to minimize the total completion time of the jobs in first set ξ_1 and for the jobs in second set ξ_2 we want to minimize the total number of tardy jobs. The processing time of the jobs in set ξ_1 and sets ξ_2 are represented by p_j^1 and p_j^2 , respectively. Similarly the due dates for the jobs in the first set and the second set can be denoted by d_j^1 and d_j^2 respectively. However, for the purpose of the objectives considered herein, due dates are only relevant for the jobs in second set.

This problem can be denoted as $1|inter|ND(\sum C_j, \sum U_j)$ using the Graham et al. [1979] notation. Clearly the notation highlights the interference between the job sets and $ND(\sum C_j, \sum U_j)$ indicates that we are looking to find the *non-dominated* (or *Pareto*

optimal) points for this problem. The non dominated point will help a decision maker to determine the tradeoffs between the interfering sets of jobs competing for the same resource.

A solution X^* is Pareto optimal or non-dominated if there exists no other solution $X \in S$ for which $z_1(X) \leq z_1(X^*)$ and $z_2(X) \leq z_2(X^*)$ where at least one of the inequalities is strict. Jazzkiewicz [2003] describes methods for evaluating the performance of multi-objective heuristics.

3 Structure of the Non Dominated Solutions

If considered independently, the single machine problems are easy to solve. Sorting the jobs in non decreasing order of processing times solves the problem of $1||\sum C_j$ while the polynomial time Moore's algorithm solves the problem of $1||\sum U_j$. However, the complexity of these performance measures with interference is yet not known. There are a few important observations / properties regarding the non dominated solution for interfering job sets and with these objectives that can be observed in the following lemmas and help further explore the structure of the non dominated solutions.

Lemma 1: There always exists a non-delay schedule for all the strongly non dominated points on the Pareto optimal front.

Lemma 2(a): For all the strongly non dominated points, there exists an optimal schedule in which jobs in the job set ξ_1 are scheduled in SPT order, Ng et al. [2006].

2(b): For all the strongly non dominated points, there exists an optimal schedule in which jobs in the job set ξ_2 that are on time are scheduled in EDD order, Ng et al. [2006].

Lemma 3: For any non dominated point, the performance criteria $\sum C_j$, for the jobs in the job set ξ_1 with preemptive scheduling remains the same as with non preemptive scheduling, provided the jobs in job set ξ_2 which caused the preemption is scheduled before the job that got preempted from ξ_1 .

We define three subset of jobs S_1 , S_2 and S_3 . Based on the above observation, for any non dominated point, the subset of jobs S_1 will contain all the jobs from ξ_1 arranged in SPT order, another subset S_2 of on time jobs from set ξ_2 which will be in EDD order and third subset S_3 of jobs that are tardy from set ξ_2 as well. The set S_3 can be arranged in any order after sets S_1 and S_2 without affecting the criteria (the interference is only between sets S_1 and S_2). This is represented in the picture below as well.

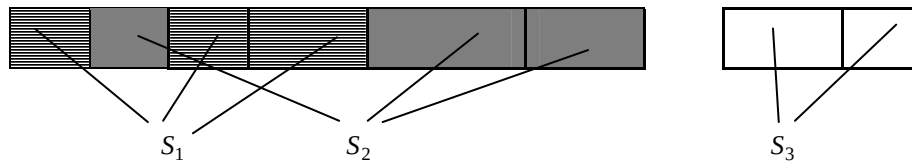


Figure 1: Structure of the non dominated point with total completion time and number tardy jobs as performance criteria

Consider the following graph which represents the structure of the efficient frontier for this problem (set of non dominated points). Let the x-axis represent the criteria of minimizing $\sum C_j$ for job set ξ_1 and y-axis represent the criteria of minimizing $\sum U_j$ for job set ξ_2 .

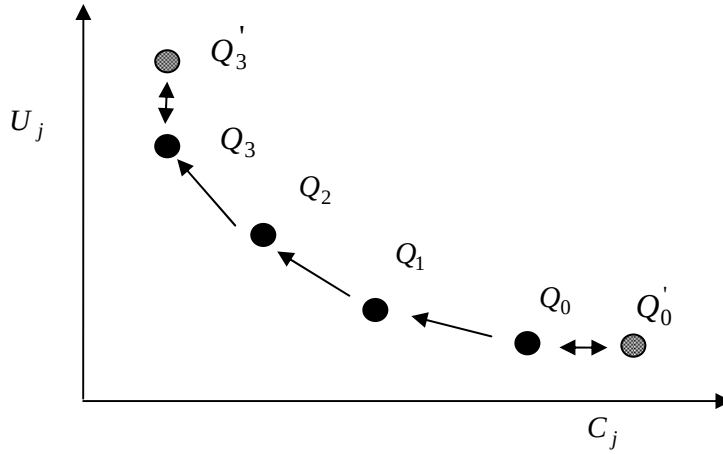


Figure2: Efficient frontier representing non dominated points for job in set C_1 and C_2 .

The points Q_0 , Q_1 , Q_2 and Q_3 in the above graph in Figure 2 represents the strongly non dominated points on the efficient frontier. The point Q_3 gives the best value of total completion time for jobs in set ξ_1 . Similarly Q_0 gives the best value of total number of jobs that are on time from set ξ_2 . The point Q'_0 is the point which is weakly non-dominated by the jobs in set ξ_1 and point Q'_3 is weakly non-dominated by the jobs in set ξ_2 . The non dominated point Q_0 can be represented by $1 | inter | \sum C_j, \sum U_j = Y_{\min}$, where $Y_{\min}$ is the minimum number of tardy jobs obtained by solving $1 || \sum U_j$ for second set without interference. Similarly non dominated point Q_3 can be represented by $1 | inter | \sum C_j = K_{\min}, \sum U_j$, where $K_{\min}$ is the minimum number of total completion time obtained by solving $1 || \sum C_j$ for first set without interference.

4 Forward SPT-EDD Heuristic

Based on the above discussion about the structure of the non dominated point and the efficient frontier of this particular problem, we can further draw an additional observation to help build a forward SPT-EDD heuristic for this problem. This observation is illustrated in the following lemma.

Lemma 4: Under the restriction that the jobs that were tardy at one non dominated point will also remain tardy at the next non dominated point as we move in the direction of improving $\sum C_j$ (jobs from set S_3 are not allowed to move back to set S_2); the one job that needs to be moved from set S_2 to set S_3 (new jobs to become tardy now) will be the one which when removed from the schedule provides the best preemptive schedule for all

the jobs in job set S_1 without moving the position of other jobs in set S_2 (hence the best improvement in the value of total completion time).

With these four distinct properties of this problem discussed so far, we present a forward SPT-EDD algorithm that can be used to generate the non dominated points for this problem. In the forward logic presented below, we start with the Moore's algorithm to determine the initial sets S_2 and S_3 . As this logic is under certain assumption and restriction, it may not yield optimal solution. In the subsequent section we do compare the computational efficiency of this algorithm with the optimal solutions from the pseudo polynomial algorithm.

The **Forward SPT-EDD algorithm** can be summarized in the following steps (here we start from point the initial point $1 | inter | \sum C_j, \sum U_j = Y_{\min}$ and then walk through point $1 | inter | \sum C_j = K_{\min}, \sum U_j$ by moving jobs from set S_2 to S_3):

Step 1: Use Moore's algorithm to determine the minimum number of tardy jobs by considering jobs in set ξ_2 alone. The solution from Moore's algorithm will help determine the sets S_2 and S_3 . The tardy jobs are placed in set S_3 while the on time jobs will be placed in set S_2 . *Note:* This step may result in a non dominated solution that is not optimal, as the division between jobs from ξ_2 in sets S_2 and S_3 obtained by Moore's algorithm, may not result in best possible value for $\sum C_j$ for jobs in the first set.

Step2: With this initial division for set ξ_2 into sets S_2 and S_3 , a non dominated solution can be determined for interfering jobs sets S_1 and S_2 using *Lemma 2a* and *Lemma 2b*.

Step 3: First the jobs in set S_2 are arranged in the EDD order in such a way that there is no earliness for the jobs in set S_2 , except when there is an overlap between jobs within set S_2 . In case of overlap, jobs with earlier due dates are placed ahead of jobs with later due dates. Now the jobs in set S_1 are arranged in SPT order allowing preemption. We finally use the property described in *Lemma 3* to get the non preemptive schedule for this non dominated point.

Step 4: Now we move jobs from set S_2 to set S_3 , using the property described in *Lemma 4* to find the subsequent non dominated point and move in the direction which brings improvement in $\sum C_j$. *Note:* Because of the restriction made in *Lemma 4*, as we proceed along the frontier to find the subsequent non dominated points, we are not considering the jobs which were tardy and in set S_3 at earlier points on the frontier to be on time in the subsequent points. We may miss some opportunity of improving the criteria for the job set S_1 because of this. The example below illustrates this gap.

We consider an **example** with 5 jobs in each set of jobs ξ_1 and ξ_2 being represented by p_j^1 and p_j^2 . For simplicity before numbering the jobs, jobs in set ξ_1 are arranged in SPT order while the jobs in set ξ_2 are arranged in EDD order. The final sequence at any non-dominated point is divided into 3 sets: S_1 which includes all jobs from ξ_1 arranged in SPT order, S_2 which includes on time jobs from set ξ_2 and S_3 which includes tardy jobs from set ξ_2 .

For set ξ_1 : $p_1^1=1, p_2^1=2, p_3^1=3, p_4^1=5, p_5^1=5$

For set ξ_2 : $p_1^2=3, p_2^2=6, p_3^2=4, p_4^2=5, p_5^2=3$

$$d_1^2=5, d_2^2=11, d_3^2=18, d_4^2=25, d_5^2=30$$

The value of $\sum C_j$ for problem 1 $\parallel \sum C_j$ is 37, while $\sum U_j$ for problem 1 $\parallel \sum U_j$ is zero. In iteration (1) since all the jobs in set ξ_2 are on time, $S_2 = \{1, 2, 3, 4, 5\}$ and $S_3 = \{\}$. Jobs in set S_1 are arranged in SPT order while jobs in set S_2 area are arranged in EDD order. This results in point $Q_0(101, 0)$.

Time Horizon																																					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	
			1				2								3							4						5									
1		1					2				2			3			3					4				5				4					5		

Figure 3(a): In the first step jobs in S_1 are allowed to be preempted. In the second step jobs in set S_2 are moved ahead to avoid preemption of jobs in set S_1 . Note that the completion time of the jobs in S_1 remains the same.

In iteration (2), it is found that moving job #2 from set S_2 to S_3 will provide maximum improvement in $\sum C_j$, hence $S_2 = \{1, 3, 4, 5\}$ and $S_3 = \{2\}$. This gives the non-dominated point $Q_1(61, 1)$.

Time Horizon																																					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	
			1												3							4							5						2		
1	1			2			3				4				3						4				5				5						2		

Figure 3(b): In the first step job #2 from set S_2 is moved to S_3 and jobs in set S_1 are allowed to be preempted. In the second step jobs in set S_2 are moved ahead to avoid preemption of jobs in set S_1 . Note that the completion time of the jobs in S_1 remains the same.

In iteration (3) it is found that moving job #1 from set S_2 to S_3 will provide maximum improvement in $\sum C_j$, hence $S_2 = \{3, 4, 5\}$ and $S_3 = \{2, 1\}$. This gives the non-dominated point $Q_2(41, 2)$.

Time Horizon																																					
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	
															3							4				5				2						1	
1	2			3				4				3				5						4				5				2						1	

Figure 3(c): In the first step job #1 from set S_2 is moved to S_3 and jobs in set S_1 are allowed to be preempted. In the second step jobs in set S_2 are moved ahead to avoid preemption of jobs in set S_1 . Note that the completion time of the jobs in S_1 remains the same.

In iteration (4) it is found that moving job #3 from set S_2 to S_3 will provide maximum improvement in $\sum C_j$, hence $S_2 = \{4, 5\}$ and $S_3 = \{2, 1, 3\}$. This gives the non-dominated point $Q_3(37, 3)$.

Time Horizon																																				
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
1	2		3					4						5				4				5				2						1			3	

Figure 3(d): Job #3 from set S_2 is moved to S_3 and jobs in set S_1 are already in a non preemptive SPT order.

Now since the best possible value of $\sum C_j$ is 37, making more jobs tardy will not provide any further improvement in $\sum C_j$, hence will result in weakly non-dominated points: $Q_4(37, 4)$ and $Q_5(37, 5)$.

5 Computational Experiments

To test the computational efficiency of the forward SPT-EDD heuristic we generate some sample data set and compare it again the results from the pseudo-polynomial algorithm of Ng et al. [2006]. We consider cases with 20, 30, 40 and 50 jobs in each sets ξ_1 and ξ_2 and generate six problem instances for each case. We select integer processing time numbers for both the set of jobs from $\sim U[1, 20]$. The due date, d_j , of job j is an integer generated from the uniform distribution $[P(L-R/2), P(L+R/2)]$, where $P = 0.5 P_1 + P_2$ (P_1 is the sum of processing time in of jobs in set ξ_1 and P_2 is the sum of processing time of jobs in set ξ_2) and the two parameters L and R are relative measures of the location and range of the distribution, respectively. We choose different values for L and R to generate three different ranges of due dates: $[0.1 P, 0.9 P]$, $[0.5 P, 0.9 P]$, and $[0.3 P, 1.1 P]$ and generate two problem instances for each range, hence generating six problem instances in total for each number of jobs.

Table 1 summarizes the comparison of average solution quality of the algorithm over 24 instances comparing it with the optimal solution obtained from the pseudo-polynomial algorithm. For the forward SPT-EDD heuristic the average percentage gap between our solution and the Pareto optimal point is less than 1%. Although the average number of non Pareto optimal points within a solution set increases with the number of jobs in ξ_2 , it remains in the range of 20-25% of the total number of points in the solution set.

# of Jobs in Each Set	Total Number of Test Instances	Forward SPT-EDD	
		Avg. Number of Non Pareto Optimal Points	Avg. % Gap of Non Pareto Optimal Points
20	6	3.8	1.14%
30	6	7.8	0.62%
40	6	7.7	0.47%
50	6	13.8	0.30%

Table1: Summary of solution quality over 24 problem instances for the Forward SPT-EDD Heuristic.

In Table2 below we present the average computational time for the algorithm over 6 different problem instances for each number of jobs. It can be observed that the computational time for the pseudo polynomial DP algorithm grows faster with the increase in the number of jobs in both sets and the total processing time of jobs in set ξ_2 , an average of about 37 seconds for the 50-job instances. The computational time for the forward SPT-EDD heuristic is under 1 second even for the 50-job instances. The effect of computational complexity can be seen in the computation times. The pseudo polynomial algorithm has a computational complexity of

$O(n_1 n_2^2 P_2)$ while the computational complexity of the proposed forward SPT-EDD heuristic is $O(n_2^2)$.

# of Jobs in Each Set	Total Number of Test Instances	Average Run Time (Seconds)	
		Dynamic Program	Forward SPT-EDD
20	6	0.898	0.086
30	6	4.495	0.214
40	6	14.984	0.487
50	6	36.703	0.979

Table2: Summary of average computational time over 24 problem instances for the Forward SPT-EDD Heuristic.

6 Conclusion and Feature Research

The proposed polynomial heuristic does a good job of providing a (nearly) non dominated solution set; the average gap is less than 1% compared to the optimal solution. The computational experiment can be continued to see the effect of the increased run time with a larger number of jobs with the pseudo-polynomial algorithm. It can be clearly seen that this SPT-EDD heuristic performs quite well and will be useful in solving job sets each with a larger number of jobs e.g. 200 or higher. Similar approach could be adopted to solve other interfering job set problems with different performance criteria and even problems which have been classified as NP-hard. We further intend to carry out more computational results as well as explore the structure of the single machine problems with two interfering job sets with criterion of total weighted completion time and number of tardy jobs as well as total weighted completion time and maximum lateness. Further it will be interesting to see if the EDD rule will still hold for on time jobs or jobs with the criteria of minimizing maximum lateness when interfering with another job set that has the criterion to minimize total weighted completion time. As we are researching and investigating these problems in several dimensions, we will discuss additional results and findings during our conference presentation.

References

1. Agnetis, Allesandro; Mirchandani, Pitu B.; Pacciarelli, Dario and Pacifici, Andrea; "Scheduling problems with two completing agents", *Operations Research*, 52(2), 229-242, 2004.
2. Baker, Kenneth R. and Smith, J. Cole; "A multiple criterion model for machine scheduling", *Journal of Scheduling*, 6, 7-16, 2003.
3. Balasubramanian H.; Fowler J.; Keha A. and Pfund M.; "Scheduling interfering job sets on parallel machines", *European Journal of Operational Research*, In press, 2008.
4. Cheng, T.C.E.; Ng, C.T. and Yuan, J.J.; "Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs", *Theoretical Computer Science*, 362, pp 273-281, 2006.
5. Jazskiewicz, A; "Evaluation of Multiple Objective Metaheuristics", *Lecture Notes in Economics and Mathematical Systems*, 535, pp 65-89, 2003
6. Ng, C.T.; Cheng, T.C.E. and Yuan, J.J.; "A note on the complexity of the problem of two agent scheduling on a single machine", *Journal of Combinatorial Optimization*, 12, 387-394, 2006.

7. Peha, J.; “Heterogeneous-criteria scheduling: minimizing weighted number of tardy jobs and weighted completion time”, *Journal of Computers and Operations Research*, 22(10), 1089-1100, 1995.

An efficient implicit enumeration for scheduling equal-length jobs with release times on a single processor to maximize throughput

Nodari Vakhania

Abstract We suggest an $O(n^2 \log n)$ combinatorial algorithm for scheduling n equal-length jobs with release times and due dates on a single machine as to minimize the number of late jobs improving the earlier proposed dynamic programming algorithm with the time complexity $O(n^5)$.

1 Introduction

Jobs from $J = \{1, 2, \dots, n\}$ have to be assigned or scheduled on a single machine when each $j \in J$ becomes available at an integer *release time* r_j and has an integer *due date* d_j which is the desired time to complete j . Each job needs an integer *processing time* p on the machine. A *schedule* assigns to every job j a time interval on the machine with the length p starting no earlier than at time r_j so that there is no intersection between intervals of different jobs; i.e., the machine may handle at most one job at a time. A job is *late* (*on time*, respectively) if it is completed after (at or before, respectively) its due date. Our objective is to minimize the number of late jobs. Due to this objective function, we may assume that every job may potentially be completed by its due date, i.e., $r_j + p \leq d_j$, for each j (then we say that job release times and due dates are *agreeable*).

Our problem is commonly abbreviated as $1/p_j = p, r_j / \sum U_j$ (U_j is a 0-1 function taking value 1 iff job j is late). Its version with arbitrary job processing times $1/r_j / \sum U_j$ is known to be strongly NP-hard. The preemptive version $1/pmtn, p_j = p, r_j / \sum U_j$ can be solved in time $O(n \log n)$ with an (off-line) dynamic programming algorithm suggested by Lawler [4] for a more general problem with job weights, a special case of $1/pmtn, r_j / \sum w_j U_j$ in which the jobs can be ordered so that $r_1 \leq r_2 \leq \dots \leq r_n$, $p_1 \leq p_2 \leq \dots \leq p_n$ and $w_1 \geq w_2 \geq \dots \geq w_n$ (an alternative on-line algorithm for $1/pmtn, p_j = p, r_j / \sum U_j$ with the same time complexity was suggested in [7]). Other variations of the single-machine version are known to be polynomial. For arbitrary jobs but without release times, the problem is also easy to solve (see Moore [5] and Lawler [4]).

Science faculty, State University Morelos, Mexico.
Partially supported by CONACyT grant 48433.
E-mail: nodari@uaem.mx

Our problem was earlier dealt with by Carlier [1]. His algorithm is sub-optimal (see [2] for the details). The first optimal polynomial-time (dynamic programming) algorithm was proposed by Chrobak et al. [2] with the time complexity $O(n^5)$. We improve this result to $O(n^2 \log n)$. Our direct combinatorial algorithm uses the Earliest Due-date heuristic (ED-heuristic) for schedule generation. The algorithm creates $O(n)$ scheduling times to each of which a partial ED-schedule corresponds (one constructed by ED-heuristic). The algorithm incorporates a kind of backtracking, hence the partial ED-schedules are represented as nodes of a solution tree T . A special kind of analysis (based on the so-called *behavior alternatives*) of each next generated schedule is carried out to determine which is to be the next enumerated schedule and whether an optimal schedule was already generated.

2 Some basics

We shall exclusively deal with ED-schedules in which all jobs are completed on time: if a job is late then it can be appended arbitrarily late to such a schedule (without affecting the objective function). From now on, we shall call a schedule *feasible* if all jobs are completed on time in it. We shall respectively refer to a job set as feasible if there is a feasible schedule including all jobs from the set. We will say that job j *conflicts* with job k if there is no feasible job set that contains both jobs: if j starts at r_j and k immediately follows j then k is late, and vice-versa. We denote by $|S|$ the number of jobs (completed on time) in a schedule S , and by $||S||$ the completion time of the latest scheduled job (the makespan) in S . We will use S for both, a schedule and the corresponding set of jobs.

Iteratively, among all released jobs by the current scheduling time t , the ED-heuristic schedules a job with the smallest deadline (ties being broken arbitrarily). Initially, $t = \min\{r_l | l \in J\}$; each successive value of t is either the completion time of the latest scheduled so far job (i.e., the previous t plus p) or if no job is released by that time, t is the minimal job release time taken among yet unscheduled jobs. We denote by t^+ the scheduling time, the next to t and will write $t \succ t'$ if a scheduling time t occurs after scheduling time t' . As it is easily seen, if a job set is feasible then the ED-heuristic will produce a feasible schedule for this set. Since ED-heuristic leaves no machine idle time which can be avoided, it will give the minimal makespan for a job set to which it is applied.

ED-heuristic is not optimal: consider a problem instance with $p = 5$ and two jobs 1 and 2 with $r_1 = 0$, $d_1 = 11$, $r_2 = 1$ and $d_2 = 6$. The heuristic will schedule first job 1 and then job 2 at time 5 making late that job, whereas job 2 can be scheduled at time 1 (creating an idle time) followed by job 1 both jobs being completed on time.

An ED-schedule can be seen as a sequence of *blocks*; a block is a sequence of jobs scheduled in turn without any idle time (a *gap*) in between. So any gap in an ED-schedule is left outside of any block and it arises only if there is no released job which can be scheduled within that gap. We will denote by B^t the current (uncompleted) block by time t and by $\tau(B^t)$ the starting time of B^t . B^t is iteratively completed by new incoming jobs until the next incoming job starts new block. Before this happens, B^t is an *extension* of $B^{t'}$, for $t' < t$ and $\tau(B^t) = \tau(B^{t'})$.

We call a job selected by ED-heuristic at scheduling time t an *incoming job* of time t ; as we will see, it may happen that the current incoming job of time t is omitted,

hence the current scheduling time does not change and the next incoming job also occurs at time t . We will use $i(t)$ for the latest so far arisen incoming job by time t .

We shall deal with partial ED-schedules and will denote the current partial ED-schedule generated to time t by S^t . $S = S^t$ can be extended by $i = i(t)$ at time t . The resultant ED-schedule, in which i is scheduled in the interval $[t, t + p)$, is denoted by $S(+i)$. S^{t+} is always set to $S(+i)$ whenever i is scheduled on time in $S(+i)$. Suppose i is late in $S(+i)$ (our algorithm is organized in such a way that the only late job in $S(+i)$ might be i). The job set $S \cup \{i\}$ can be feasible or not. If $S \cup \{i\}$ is not feasible, some job from $S \cup \{i\}$ is to be omitted to get a feasible schedule. We will say that a job is *disregarded* (*omitted*, respectively) if it is discarded and is never again considered for the inclusion (discarded but might later again be considered for the inclusion, respectively).

Suppose $i(t)$ does not conflict with the latest scheduled job in $S = S^t$ and still cannot be scheduled on time. We wish to know whether the job set $S \cup \{i\}$ is feasible. Let us call job $e \in B^t$ with $d_e > d_i$ an *emerging job* at time t . We may have none, one or more emerging jobs at time t which will be referred to as *alternative* emerging jobs. Given an emerging job e , a derived ED-schedule denoted by S_e (a *complementary* to S schedule or *C-schedule* for short) has the following property. In S_e job e and any other job not from S is forced to be scheduled after job i (so we schedule 1 less job before i). We will say that e is *activated* for job i . We will refer to job $m \in B^t$ as *marched* by time t if it is in the state of activation in S (we may have none, one or more marched jobs in S). Among the alternative emerging jobs, one with the maximal ordinal number is called the *live* (it will commonly be denoted by l).

To actually create S_e , we first increase artificially the release time of e to r_j and that of any $j \notin S$ to $\max\{r_i, r_j\}$. Then we apply the ED-heuristic to the modified instance: none of the above jobs can then become an incoming job ahead job i . Once we activate e we again apply ED-heuristic for the yet unscheduled (non-disregarded) jobs till the next incoming job is late (we will successfully generate our destiny optimal schedule if there occurs no late job). We shall deal with the case when this late job is again i in Section 3. Otherwise, e will become an incoming job at some time $t' > t$ and will be included if it can be completed on time (we shall deal with the other case also in Section 3). If this happens, e might be activated again for some following incoming job $i(t'')$, $t'' > t'$, and this scenario might be repeated.

The set of jobs in S scheduled between the live emerging job l and i , including i , is called the *kernel* of S , denoted by $K(S)$. All kernel jobs are more urgent than any corresponding emerging job. The next statement from [6] (Lemma 1) follows from ED-heuristic and from the facts that $r_i > r_e$ and that all potential jobs which could have been become incoming jobs ahead of job i are forced to be scheduled after job i . We use $\phi(e, S)$ for the ordinal number of e in S :

Fact 1 *There arises (at least one) gap between the $(\phi(e, S) - 1)$ st scheduled job and the earliest scheduled job of $K(S)$ in S_e . Consequently, kernel jobs restart earlier. Moreover, if e is the live emerging job, then there will arise a gap immediately before the earliest scheduled job of K in S_e (hence the earliest kernel job will be scheduled at its release time).*

It will also be useful another easily seen fact from the above reference [6]:

Fact 2 *Suppose k and e are alternative emerging jobs in S with $\phi(k, S) < \phi(e, S)$. Then the starting time of the earliest scheduled job of $K(S)$ in S_k is more than or equal to*

that of the earliest scheduled job of $K(S)$ in S_e . In general, the left-shift of the kernel jobs and successive jobs in S_e is no less than that in S_k .

Note that the gaps from Fact 1 can be avoided, unlike normal (natural) gaps arisen while applying ED-heuristic to the original problem instance. We shall refer to the former gaps as the ones *imposed* by (the activation of) e . Note also that the imposed gaps are not relevant to the block definition (no imposed gap starts a new block, and any idle time within a block corresponds to an imposed gap). The imposed gap(s) arisen after the activation of e for some i will disappear if the following operations are carried out: (1) the release time of the corresponding jobs with the modified released times including e , prior to that activation is restored; (2) ED-heuristic is applied to that newly modified instance. We shall refer to these operations as the *revision* of e for i . If a job is revised for the earliest incoming job for which it was activated then it will be returned to its “natural place” and will no longer be marched. We shall use $em(i)$ for the emerging job that is in the state of activation for i (i.e., it was activated for i and has not been revised; in general, no revised job can again be activated).

We will distinguish emerging jobs scheduled after any imposed gap from those scheduled before some imposed gap in B^t . We call former jobs *normal* and latter jobs *passive* emerging jobs in S^t (in the sequel, an “emerging job” is used for either of these types). Once an emerging job is marched all emerging jobs scheduled originally before the activated emerging job become passive, and every passive emerging job yields at least one marched job. If there is no marched emerging job m then no passive one may exist. A marched emerging job may also become a passive (and then again a normal) emerging job.

A passive emerging job π is “isolated” from the following part of S^t by the imposed gap(s) arisen after the activation of the corresponding marched job(s). Because of these gaps, a passive emerging job π , unlike a normal one, cannot be beneficially activated as this cannot give the desired left-shift: if there is an (imposed) gap between jobs π and j , clearly j cannot be restarted earlier if π is rescheduled after j . Thus a passive emerging job is of no use before it is converted into a normal one. For this, all imposed gaps after π are to be eliminated, hence the corresponding marched emerging job(s) need to be revised.

Proposition 1 *Suppose the late incoming job $i = i(t)$ is not marched, and there exists no emerging job (neither normal nor passive) in S^t . Then the set $S^t(+i)$ is not feasible. Moreover, i can be disregarded.*

Proof. From the conditions it follows that the only possibility to schedule i on time is to reschedule some job $j \in B^t$ with $d_j \leq d_i$ after i or to omit such a j . j will be late if rescheduled after i , so the set $S^t \cup \{i\}$ is not feasible and either i or j is to be omitted. Clearly, the implications of each of these options are to be seen on scheduling times $t' \succ t$ while considering the possibility of the inclusion of $i(t')$. In particular, t' is to be the minimal possible given the number of the earlier included jobs. By Fact 2 and due to $p_i = p_j$, this will be provided by the omission of i . $\square$

By activating an emerging job more urgent jobs are given a chance to be (re)started earlier. A marched job will remain unscheduled as long as yet unscheduled jobs are more urgent and no idle time is created by ED-heuristic. A marched job m is either already included into S^t or is $i(t)$ or neither it belong to S^t nor it is $i(t)$. In the latter case m will be referred to as *hidden* by time t . We may have none, one or more hidden jobs at time t . The same job may become hidden and again included into the current schedule

two or more times. In this case this job should have been activated the same number of times for different incoming jobs (each activation yielding its own imposed gaps).

3 Behavior alternatives and their analysis

Our further decisions for the case when $i = i(t)$ is late are based on the case-analysis of our behavior alternatives, the first of which is introduced now. We will say that an *instance of alternative (a)* (IA(a) for short) occurs at time t if i is the late incoming job not only in $S(+i)$ but also in $S_l(+i)$ (as always, $S = S^t$ and l is the live emerging job in S).

Proposition 2 *The incoming job $i(t)$ can be disregarded if there occurs IA(a) at time t .*

Proof. First we note that $K \setminus \{i\} = \emptyset$ is not possible. Indeed, this would mean that l immediately precedes i in $S(+i)$. Then there is a gap immediately before job i in S_l , i.e., i starts at its release time and hence cannot be late (release times and deadlines are agreeable). So, suppose $K \setminus \{i\} \neq \emptyset$. As we have observed in Fact 1, there is a gap before the earliest scheduled job of K in S_l , i.e., the first scheduled job of K starts at its release time in S_l . Hence, the only possibility to schedule i earlier (than it was scheduled in $S_l(+i)$) is to reschedule a job from $K \setminus \{i\}$ after i . But by the definition of job l , all jobs from K have a due date no greater than d_i . Hence, any of these jobs will be late if rescheduled after i and i can be disregarded, similarly as in Proposition 1. $\square$

Due to Proposition 2, assume for the rest of this section that no IA(a) occurs. Suppose again $i = i(t)$ is late and there exists no emerging job in S^t ; we call a *marshalled job m squeezed* if either $m = i$ or $m \in B^t$ (so m is not hidden and it is not an emerging job in S^t). Intuitively, squeezed jobs are these marshalled jobs whose due-date turns out to be insufficiently small; they did some work but they got “old” (the younger generation with a larger due-date is needed).

Now we give a brief description of our first ALGORITHM I, which works under the assumption that at none of the scheduling times a squeezed/hidden job arises. If during the execution of ED-heuristic on none of the scheduling times a late job occurs then the schedule produced by ED-heuristic is clearly optimal. Otherwise, we omit a late incoming job $i(t)$ if one of the Propositions 1, 2 at time t apply, and we continue with the next scheduling time. Otherwise, if there exists a normal emerging job at time t then we activate the live emerging job l (i.e., we set $S^{t+} := S_l^t(+i)$). Otherwise, as Proposition 1 did not apply, there must exist a squeezed or a hidden job in S^t . ALGORITHM I has no predetermined actions for this case with which we deal with below. The reader should have no difficulty in completing our observations to a proof of this lemma:

Lemma 1 *ALGORITHM I generates an optimal schedule if no squeezed/hidden job during its execution arises.*

If during the execution of ALGORITHM I a squeezed/hidden job at time t occurs then there might be necessary to reconsider some activation(s) carried out before time t rather than just discard job $i(t)$. For example, we may have yet inactivated (passive) emerging job with a small enough due-date that may successfully substitute the activation(s) of the squeezed job(s). We study such circumstances in what follows.

We will say that an *instance of alternative (b1)* (IA(b1)) at time t occurs if the late incoming job $i = i(t)$ is squeezed (this case was not dealt with in Proposition 1). We also distinguish a variation of IA(b1). We again have a squeezed job m in S^t but it is not now $i(t)$, and again there is no normal emerging job in S^t ($i(t)$ being late). As for IA(b1), we may have a passive emerging job in B^t or may not. If there exists no passive emerging job then let $m_1, m_2, \dots, m_l, l \geq 1$, be the marched jobs in B^t in the order as they were activated. Otherwise let π be the live (the latest scheduled) passive emerging job in S^t , and let $m_1, m_2, \dots, m_l, l \geq 1$, be the intermediate marched jobs (ones activated already once π was scheduled) also ordered according to their activation times (π , in general may also be a marched job). Observe that none of m_l s is an emerging job in S^t and these jobs have imposed the gaps following job π in S^t (they are in the state of activation in S^t). Moreover, π cannot be (beneficially) activated unless all these intermediate marched jobs are revised. Each m_l is squeezed unless it is hidden. If all m_l s are squeezed (none of them is hidden) then we will say that an *instance of alternative (b2)* (IA(b2)) at time t occurs. Otherwise (at least one of m_l s is hidden) we will say that an *instance of alternative (h)* (IA(h) for short) at time t occurs. Recall that by our assumption in none of the schedules IA(a) occur. In other words, every incoming job for which one of the squeezed/hidden emerging job was activated has been included in the corresponding ED-schedule.

While for IA(b1/b2) the activation of a passive emerging job may potentially be beneficial, this is not the case for IA(h) (note that this claim does not apply to a normal emerging job, i.e., a normal emerging job can be beneficially activated even in the presence of a hidden job). Indeed, let among all hidden m_l s h be the latest activated one, and γ be the latest imposed gap by h . Then clearly, the set constituted by the jobs scheduled after γ in S^t and job $i(t)$ is not feasible; then one of these jobs is to be omitted in any feasible schedule. Similarly as in Proposition 1, we easily obtain that $i(t)$ can be omitted as long as h remains in the state of activation. We return to this point with more details later in this section.

For IA(b1/b2), an alternative passive emerging job π may be beneficially used as a substitution for the corresponding squeezed job(s). In general, more than one (alternative) passive emerging job may exist. Each such an emerging job may potentially be converted into a normal one only if the corresponding intermediate marched jobs are revised. Alternative emerging jobs can always be tried in the decreasing order of their ordinal numbers by on Fact 2. Thus at each scheduling time t with the late $i(t)$, among all alternative emerging jobs the live emerging job is always activated first. Likewise, whenever there occurs a need in an alternative emerging job (IA(b1/b2) arises), among all yet untried emerging jobs the latest scheduled one is activated the next. In other words, the alternative emerging jobs are tried in the decreasing order of their ordinal numbers: each next activated alternative (former passive) emerging job ϵ has a smaller ordinal number than the previously activated (and squeezed) one, say e , and besides $d_\epsilon < d_e$. Recall that before the earliest (live) passive emerging can be activated all the imposed gaps after π have to disappear, i.e., every intermediate m_l is to be revised. In case m_l was activated several times, all these activations will be reverted if m_l is revised for the earliest incoming job for which it was activated. Note that no yet untried passive emerging job could have been scheduled in between these activations of m_l , as otherwise such a job would have been activated ahead m_l .

The live passive emerging job π can be converted into a normal emerging job at time t if the intermediate m_l s are revised and ED-heuristic is applied (then π becomes the latest scheduled yet untried normal emerging job to be activated next). For IA(b1),

the squeezed $i(t)$ is to be revised as well. The release times of all these squeezed jobs and that of job π are modified in one turn and ED-heuristic to the modified problem instance is applied. We shall refer to these our actions as the *substitution* initiated by π . Note that while π is activated for the current late incoming job it is also substituting the earlier activations of the m_i s (and $i(t)$ in case of IA(b1)).

Thus whenever a former passive emerging job becomes squeezed (IA(b1/b2) occurs), the substitution initiated by the live passive emerging job is carried out. In this way, a series of repeated instances of alternative (b1/b2) may occur (each new such an instance will arise as long as the next passive emerging job exists). Consider the k th element from the series, $k \geq 2$, occurring at time t . The left-shift for the jobs included after the imposed gaps for the k th element might be less than that for the $(k - 1)$ th instance due to Fact 2. Hence, these jobs may start later than they were started before the k th instance (in S^t) and some may become late. We wish to include the late job $i(t)$ on time, whereas the number of jobs scheduled before $i(t)$ is to be one less than $|S^t|$: it cannot be $|S^t|$ as now a new emerging job j is to be included after $i(t)$, and it should not be less than $|S^t| - 1$ (as otherwise we will be losing an extra job). Assume the left-shift corresponding to the k th element from the series is “not enough”: either $i(t)$ could not have been included or the number of jobs included before $i(t)$ is less than $|S^t| - 1$. Then we will say that the substitution initiated by job j is *weak*, where j is the live passive emerging job of time t (the one initiated the substitution corresponding to the k th element).

Suppose the substitution initiated by j is weak, and we substitute j by an earlier scheduled (yet untried) alternative emerging job still hoping to recuperate the late job $i(t)$. Then due to Fact 2 this new substitution will also be weak and hence nothing is to be gained from it. Thus all yet untried potential substitutions can be neglected whenever the latest accomplished substitution is weak:

Proposition 3 *All yet untried potential substitutions can be neglected if the substitution initiated by a former passive emerging job is weak.*

An exhaustive instance of alternative (b). We introduce our last behavior alternative, a special case of IA(b1/b2). It delineates a stage in the algorithm when the omission of one extra job (either the current incoming job or some job from B^t) becomes unavoidable, as all the potentially useful emerging jobs have already been tried. There occurs an instance of one of the alternatives (b1/b2) or the latest accomplished activation is weak: either there existed no passive emerging job for that IA(b1/b2), or all the (former passive) alternative emerging jobs have already been tried (with the outcome of IA(b1/b2)) or the latest accomplished substitution was weak (see Proposition 3). Then we will say that there occurs an *exhaustive instance of alternative (b)* at time t (EIA(b) for short).

Consider the earliest instance of alternative (b1/b2) from the respective series of instances of alternative (b1/b2) ending with EIA(b). If there exists no passive emerging job then the series consists of this single instance of alternative (b1/b2); otherwise, the number of instances (b1/b2) in the series is bounded by the corresponding number of the passive emerging jobs. In general, we may have two outcomes for a series consisting of one or more occurrences of IA(b1/b2): it either completes with EIA(b) (the unsuccessful outcome), or the latest activated/substituted emerging job turns out to be an appropriate candidate, i.e., neither it becomes squeezed nor the corresponding

substitution is weak, hence no EIA(b) for the corresponding series occurs (the successful outcome). In the latter case another EIA(b) may occur only within some succeeding block.

Now we discuss in more details the implications of exhaustive instances of alternative (b) assuming that EIA(b) at time t arises. We use $ED(S)$ for the ED-schedule generated for job set S , and we denote by $I(B^t)$ the set of all the former late incoming jobs in block B^t . Observe that each $i \in I(B^t)$ has been successfully processed within B^t , i.e., an emerging job for i was activated (with the non-weak outcome). From the definition of EIA(b) and Propositions 1 and 3 it follows that the job set $B^t \cup \{i(t)\}$ is not feasible. Hence, a feasible schedule cannot contain all jobs from this set, i.e., one of these jobs is to be omitted. We claim that among all such candidate jobs $j \in B^t \cup \{i(t)\}$ with $|ED(B^t \cup \{i(t)\} \setminus \{j\})| = |B^t|$, one yielding the minimum $||ED(B^t \cup \{i(t)\} \setminus \{j\})||$ can be omitted. Indeed, $|B^t|$ is the maximal number of jobs in $B^t \cup \{i(t)\}$ that can be feasibly scheduled. By the definition of EIA(b) and Proposition 3, no job from B^t can be beneficially activated at any scheduling time $t' \succ t$. Then no job from $B^t \cup \{i(t)\}$ is to be inserted at or after scheduling time t' (see Proposition 1) and our claim clearly follows. Now, by omitting any j we will liberate an interval with the same length p . The later a job is scheduled the more might be the left-shift for successively scheduled jobs in the case it is omitted (Fact 2). Hence, job $i(t)$ is the best possible candidate in this sense. However, if we omit job $j \in I(B^t)$ revising the corresponding emerging job $em(j)$ we may gain an extra space as the gaps imposed by this activation will be used by the included jobs. This gives a potential for the left-shift to the jobs have been scheduled after these gaps. The following lemma is now apparent:

Lemma 2 (A sub-block optimality Lemma) *Job set $B^t \cup \{i(t)\}$ is not feasible if EIA(b) at time t occurs, hence at least one job from the set is to be omitted. Furthermore, among all such candidate jobs $j \in B^t \cup \{i(t)\}$ with $|ED(B^t \cup \{i(t)\} \setminus \{j\})| = |B^t|$, one yielding the minimum $||ED(B^t \cup \{i(t)\} \setminus \{j\})||$ can be omitted. This minimum is reached when $j \in I(B^t) \cup \{i(t)\}$ and when $em(j)$ is revised for j , for $j \in I(B^t)$, and by omitting j , for $j = i(t)$.*

Based on this lemma a procedure to find a job in $I(B^t) \cup \{i(t)\}$ minimizing the above magnitude is straightforward. The procedure will have no special cost within the framework of the whole algorithm (though its direct application yields the running time $O(\nu n \log n)$, where ν is the number of jobs in $I(B^t)$). Let us first give some details of this procedure called *SCHEDULE_SUBBLOCK*(t). We aim to find a $j \in I(B^t) \cup \{i(t)\}$ with the minimum $||ED(B^t \cup \{i(t)\} \setminus \{j\})||$ provided that $|ED(B^t \cup \{i(t)\} \setminus \{j\})| = |B^t|$. Either $j \in I(B^t)$ or $j = i(t)$. By omitting a job from $I(B^t)$ we give a possibility for the left-shift to the succeeding jobs from B^t and $i(t)$, while the above minimum provides the earliest starting for the jobs which will be scheduled after time t (given the number of jobs included by time t). Let j' be any job from $I(B^t)$ succeeding j in S^t . We wish to keep both j' and $em(j')$ scheduled. Moreover, we have to revise $em(j')$ for j' in order to give a chance to the succeeding jobs to be left-shifted. (Remind that even though job j is omitted, such left-shift cannot occur as long as the corresponding imposed gap(s) by $em(j')$ remain; these gaps will disappear if $em(j')$ is revised for j' .) Let τ be the starting time of $em(j')$ prior to its activation for j' (i.e., that in $S^{t'}$, with $j' = i(t')$). Roughly, if $em(j')$ is not released early enough before time τ then it cannot be sufficiently left-shifted within the liberated space due to the omission of job j . In particular, this left-shift should be more than p , as otherwise we may provide a not less left-shift for the rest of the succeeding jobs by just revising $em(j')$ for j' , omitting

j' instead of j . Hence, it is apparent that job j does not need to be omitted unless $r_{em(j')} < \tau - p$, for every j' succeeding j in S^t . For similar reasons, the omission of j will be of no benefit if the (total) length of the imposed gap(s) by $em(j)$ is no more than that of $em(j')$, for all j' .

It is apparent now why *SCHEDULE_SUBBLOCK*(t) processes the jobs from $I(B^t)$ in the decreasing order of their start times in S^t considering the latest scheduled candidates first. Note that job $i(t)$ need not be processed as the corresponding partial ED-schedule is the same as B^t , so $||ED(B^t \cup \{i(t)\} \setminus \{j\})|| = ||B^t|| = t$. The procedure stops when the above described conditions hold for all yet unprocessed j s or all candidate jobs have already been processed. Note that *SCHEDULE_SUBBLOCK*(t) not only determines job j but also arranges the corresponding portion of the schedule (minimizing $||ED(B^t \cup \{i(t)\} \setminus \{j\})||$).

Observe that the number of jobs in $I(B^t)$ is the same as that of the corresponding marched jobs unless some $em(j)$ is activated for two or more times for different jobs from $I(B^t)$ (in row). In this particular case, when considering the possibility of the omission of each of these jobs from $I(B^t)$, $em(j)$ is revised for that selected job.

Consequences of IA(h). Consider the earliest IA(h) occurring within our current block B^t , $h_1, h_2, \dots, h_k$ being the corresponding hidden jobs at time t . Since each h_i becomes hidden while all $h_1, h_2, \dots, h_{i-1}$ still remain hidden, $d_{h_1} > d_{h_2} > \dots > d_{h_k}$. We remind that $i(t)$ is late and there exists no normal emerging job in S^t (that could potentially become another hidden job). We have also seen that the set of jobs scheduled after the imposed gap(s) by (any) h_i together with job $i(t)$ is not feasible, and that $i(t)$ is omitted (but not disregarded) at time t . We shall refer to each such an omitted $i(t)$ as a *dependent* job (this definition extends, in general, to any instance of alternative (h)). Each new dependent job $i(\tau) \in B^t$ is omitted at time τ . Hence the current τ does not change and there may occur more than one dependent job at time τ . As we will see now, dependent jobs delineate an interval within which there are more candidates to be included than can actually be (on time) included. Hence we have some freedom in the selection of a subset of jobs to be included.

Let $i(h_i)$ be the (former late incoming) job for which h_i was activated before it got hidden ($i = 1, 2, \dots, k$), and let $s(h_i)$ be the start time of h_i before it was activated and got hidden. We have the gap(s) imposed by each h_i between time $s(h_i)$ and job $i(h_i)$ in S^t (with the total length less than p); if any job is scheduled within such a gap then $i(\tau)$ cannot be included on time and will be omitted. This might be a correct thing to do as we will see now. When we have activated h_i for $i(h_i)$ we hoped to keep jobs $i(h_i)$, h_i together with the next incoming jobs scheduled. However, this hope disappears when a corresponding dependent job occurs. The set of jobs scheduled between $s(h_i)$ and t (including $i(h_i)$), together with the dependent jobs, is not feasible: too many jobs to be included within that interval. It might be better to omit $i(h_i)$ and include a dependent job as the corresponding imposed gaps might be reduced or eliminated at all (by scheduling job(s) released within or before these gaps). This, in turn, gives a potential to (re)start later scheduled jobs earlier and hence to increase the number of the included jobs.

Roughly, if we have enough dependent jobs then we can include one dependent job instead of some $i(h_i)$ s in case this yields the reduction of the imposed gaps between times $s(h_1)$ and t . If there are enough jobs released early enough so that all the imposed gaps disappear then we may clearly assert that the resultant ED-portion is optimal, that is, it is to be a part of the destiny optimal schedule.

We next specify how a set of dependent jobs can naturally be associated with some hidden jobs. Recall that we had k hidden jobs $h_1, h_2, \dots, h_k$ when the earliest instance of alternative (h) within the current B^t at time t has occurred: $i(t)$ is the first arisen late incoming job for which no normal emerging job exists, we have called it dependent job (given that there is at least one hidden job). Once $i(t)$ is omitted, there may occur repeated IA(h) at time t , i.e., the next incoming job may again turn out to be a dependent job. A similar picture might be repeated at time t or at some succeeding scheduling time.

Let $t' \succ t$ be the earliest scheduling time such that job h_k is no more hidden, i.e., $i(t') = h_k$ (as $d_{h_1} > d_{h_2} > \dots > d_{h_k}$, h_k will be the first of h_ι s that will (again) become an incoming job). Further, let $D(t')$ be the set of dependent jobs arisen in between times t and t' . No $j \in D(t')$ can be released before time $s(h_k)$. Indeed, by definition of j , $d_j < d_{h_k}$ and hence ED-heuristic would include j instead of h_k if j were released by time $s(h_k)$. So j can potentially be included only after time $s(h_k)$, hence cannot “substitute” any $i(h_\iota)$, $\iota = 1, 2, \dots, k-1$. Hence jobs from $D(t')$ can be included only within the interval $[s(h_k), t']$, instead of job $i(h_k)$ (reducing or eliminating at all the corresponding imposed gaps). So $D(t')$ is naturally associated with h_k (and with none of the other hidden jobs). Later in the formal description we will also use $D(h_k)$ for this set, and we call its jobs dependent *on* h_k . As we have already seen, $D(h_\iota) = \emptyset$ for $\iota < k$, i.e., to some former hidden jobs no dependent job may correspond.

Either (i) $h_k = i(t')$ is included at time t' or (ii) it is late (hence IA(b1) at time t' occurs). Consider first case (i): job h_k is no more hidden and it got included. $D(h_k)$ cannot grow any more and we may try to reschedule the interval $[s(h_k), t']$. This time, we do not need to force job $i(h_k)$ to be included, we know that this job might be omitted and a corresponding dependent job included instead. We restore the release times of all the jobs except job h_k that were artificially modified (increased) during the activation of h_k for $i(h_k)$ (such a job might well be one from $D(h_k)$); now these jobs will no more be forced to be scheduled after $i(h_k)$ and hence might be rescheduled earlier and the imposed gaps by h_k can respectively be reduced/eliminated. We call this the *weak revision* of job h_k . Note that the current release time of job h_k itself is not modified as we still hope to include both, h_k and some dependent job from $D(t')$ (thus the name for such a revision). At the same time such an option is not to be realized if there still remains an imposed gap by h_k once it is weakly revised and h_k gets squeezed at some posterior scheduling time (in the case the (normal) revision of h_k is clearly better as all the imposed gaps will be eliminated). Thus we came to the following proposition:

Proposition 4 *The portion of ED-schedule obtained after the weak revision of h_k is optimal if h_k , once revised, does not get squeezed.*

The case not covered in this proposition (h_k becomes squeezed) and case (ii) above are similar: in the former case we have IA(b2), and in the latter case IA(b1) occurs (hence we revise job h_k for $i(h_k)$ as earlier specified for these instances).

Once h_k is scheduled at time t' and processed as above, another hidden job h_{k+1} with a non-empty $D(h_{k+1})$ may occur, and so on. Observe at the same time that at most one yet unprocessed hidden job h with a non-empty $D(h)$ for each scheduling time t may exist. We shall refer to that job as the *heading* (hidden) job of time t .

4 Summarizing the algorithm

In the formal description of our algorithm t is the current scheduling time with $S = S^t$ being the current schedule and $i = i(t)$ the corresponding incoming job. The values of t and S are updated according to the description. $\text{ACTIVATE}(l)$ activates the live emerging job l for the late incoming job.

ALGORITHM II

initial settings:

$S := \emptyset; t := \min\{r_j | j \in J\}$

Procedure MAIN

IF no unconsidered job is left THEN *output* S ; *halt*

IF $i = i(t)$ is on time in $S(+i)$ THEN $(S^{t^+} := S(+i))$; weakly revise i if it is
the heading job)

ELSE $\{i \text{ is late in } S(+i)\}$

begin $\{i \text{ is late}\}$

IF i conflicts with the latest scheduled job in S THEN omit i

IF there exists no emerging job and i is not marched in B^t THEN disregard i

IF there exists a normal emerging job THEN $\text{ACTIVATE}(l)$

IF there occurs IA(a) THEN disregard i and revise l for i

IF there occurs IA(h) THEN update $D(h)$ for the heading job h

IF there occurs IA(b1/b2) and there exists the live passive emerging job π

THEN carry out the substitution initiated by π if it is not weak

ELSE $\{ \text{there occurs EIA(b)} \}$ $\text{SCHEDULE\_SUBBLOCK}(t)$

end $\{i \text{ is late}\}$

end $\{ \text{main} \}$.

Theorem 1 *ALGORITHM II produces an optimal schedule in time $O(n^2 \log n)$.*

Proof. The schedule generated by the algorithm is optimal if the ED-heuristic schedules each incoming job on time. The time complexity in this case is $O(n \log n)$ (that of the ED-heuristic). In general, the number of scheduling times is $O(n)$. Indeed, from any scheduling time t to time t^+ , either one job is scheduled, or one job is omitted or a substitution initiated by the next passive emerging job is carried out. Each such a job may initiate a single substitution whereas the total number of emerging jobs is $O(n)$. Hence, the total number of scheduling times is $O(n) + O(n) + O(n) = O(n)$. At any scheduling time t , we either add or omit a job (with no extra cost), or activate the live emerging job or carry out a substitution with a brutal cost of $O(n \log n)$ (that of ED-heuristic), or call procedure $\text{SCHEDULE\_SUBBLOCK}(t)$. The latter implies the application of ED-heuristic to the modified problem instance from the (earlier) scheduling time, the next to time $r_{em(j)}$, for every $j \in I(B^t)$. This yields a single extra application of ED-heuristic for some earlier considered scheduling times, hence it has no extra cost. So the overall time complexity is $O(n^2 \log n)$.

The soundness part follows from the already established facts. We give a sketch. A late incoming job $i(t)$ can be disregarded if it is not marched and there exists neither normal nor passive emerging job in S^t (no marched job in B^t) By Proposition 1. By Fact 2, if there exists a normal emerging job in S^t then the live one is to be tried first; more generally, alternative emerging jobs are to be tried in the decreasing order of their ordinal numbers. If Proposition 1 does not apply and there exists no (yet untried) normal emerging job in S^t then one of the behavior alternatives will occur (whenever i is late). So assume there exists a normal emerging job at time t and we activate the live emerging job l . If in the resultant C-schedule i still cannot be included on-time,

i.e., IA(a) occurs, i is disregarded by Proposition 2. It remains to consider instances of alternatives (b1/b2), (h) and exhaustive instances of alternative (b). As to IA(h), as we have seen in the previous section, each heading job that doesn't become squeezed is to be weakly revised (consequences of IA(h)). In the rest of the cases IA(b1/b2) or/and EIA(b) occurs. For EIA(b) procedure SCHEDULE_SUBBLOCK(t) produces a segment of the destiny optimal schedule by Lemma 2. Finally, as we have already seen for IA(b1/b2), a substitution initiated by the next (yet untried) alternative emerging job is to be carried out. $\square$

5 Acknowledgements

The author is grateful to Christoph Durr for his attention and comments.

References

1. J. Carlier. "Problèmes d'ordonnancement à durées égales". *QUESTIO* 5,219-228 (1981)
2. M. Chrobak, C. Durr, W. Jawor, L. Kowalik and M. Kurowski. "A note on scheduling equal-length jobs to maximize throughput". *Manuscript* (2004)
3. E.L. Lawler. "A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs". *Annals of Operations Research* 26, 125-133 (1990)
4. E.L. Lawler. "Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the tower of sets property". *Mathematical Computer Modelling* 20, 91-106 (1994)
5. J.M. Moore. "An n job, one machine sequencing algorithm for minimizing the number of late jobs". *Management Science* 15, 102-109 (1968)
6. N. Vakhania. "A better algorithm for sequencing with release and delivery times on identical processors". *Journal of Algorithms* 48, 273-293 (2003)
7. N.Vakhania. "Fast algorithms for preemptive scheduling of equal-length jobs on a single and identical processors to minimize the number of late jobs". *Int. J. of Mathematics and Computers in Simulation* 1, p.95-100 (2008)

Isomorphic coupled-task scheduling problem with compatibility constraints on a single processor

Gilles Simonin · Benoît Darties · Rodolphe Giroudeau · Jean-Claude König

Abstract The problem presented in this paper is a generalization of the usual coupled-tasks scheduling problem in presence of compatibility constraints. The reason behind this study is the data acquisition problem for a submarine torpedo. We investigate a particular configuration for coupled-tasks (any task is divided into two sub-tasks separated by an idle time), in which the idle time of a coupled-task is equal to the sum of durations of its two sub-tasks. We prove $\mathcal{NP}$ -completeness of the minimization of the schedule length, and we show that finding a solution to our problem amounts to solving a graph problem, which in itself is close to the minimum-disjoint path cover (min-DCP) problem. We design a $\left(\frac{3a+2b}{2a+2b}\right)$ -approximation, where a and b (the processing time of the two sub-tasks) are two input data such as $a > b > 0$, and that leads to a ratio between $\frac{3}{2}$ and $\frac{5}{4}$. Using a polynomial-time algorithm developed for some class of graph of min-DCP, we show that the ratio decreases to $\frac{1+\sqrt{3}}{2} \approx 1.37$.

1 Introduction

In this paper, we present a scheduling problem of coupled-tasks subject to compatibility constraints, which is a generalization of the scheduling problem of coupled-tasks first introduced by Shapiro [18]. This problem is motivated by the problem of data acquisition in a submarine torpedo. The aim amounts to treating various environmental data coming from sensors located on the torpedo, that collect information which must be processed on a single processor. A single acquisition task can be described as follows: a sensor of the torpedo emits a wave at a certain frequency (according to the data that must be collected) which propagates in the water and reflects back to the sensor. This acquisition task is divided into two sub-tasks: the first sends an echo, the

Gilles Simonin · Rodolphe Giroudeau · Jean-Claude König
LIRMM UMR 5506, rue Ada, 34392 Montpellier Cedex 5 - France
E-mail: firstname.lastname@lirmm.fr

Benoît Darties
Grenoble Informatics Laboratory (LIG) CNRS, Grenoble-INP 681 rue de la passerelle, BP72
38402 Saint Martin d'Heres Cedex, France
E-mail: benoit.darties@imag.fr

second receives it. Between them, there is an incompressible idle time which represents the spread of the echo under the water. Thus acquisition tasks may be assigned to coupled-tasks.

In order to use idle time, other sensors can send more echoes. However, the proximity of the waves causes disruptions and interferences. In order to handle information error-free, a compatibility graph between acquisition tasks is created. In this graph, which describes the set of tasks, we have an edge between two compatible tasks. A task is compatible with another if at least one of its sub-tasks can be executed during the idle time of another task. Given a set of coupled-tasks and such a compatibility graph, the aim is to schedule the coupled-tasks in order to minimize the time required for the completion of all the tasks.

1.1 Problem formulation

According to the torpedo problem, a task may be started during the idle time of a running task if it uses another frequency, is not dependant on the execution of the running task, or does not require to access the resources used by the running tasks. Formally, we say that two tasks A_i and A_j are compatible if and only if we can execute at least a sub-task of A_i during the idle time of A_j (see fig. 1(b)). On the other side, some tasks cannot be compatible due to previously cited reasons. The compatibility graph G_c summarizes these compatibilities constraints, whose edges $E(G_c)$ represent all pairs of compatible task.

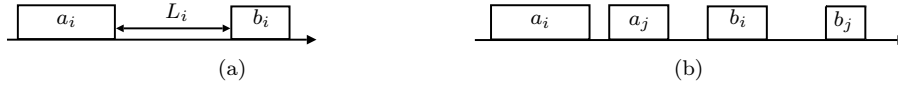


Fig. 1 Understanding a single coupled-task and two compatible coupled-tasks.

For a graph G , we note $V(G)$ the set of its vertices and $E(G)$ the set of its edges. The cardinality of both sets are noted $n = |V(G)|$ and $m = |E(G)|$. In the following, we will call *path* a non-empty graph $C = (V, E)$ of the form $V = \{x_0, x_1, \dots, x_k\}$ and $E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\}$, where the x_i are all distinct. The number of edges of a path corresponds to its length. The input of the general problem is described with a set $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$ of coupled-tasks and a compatibility graph $G_c = (\mathcal{A}, E(G_c))$. Using the notation proposed by Shapiro [18], each task $A_i \in \mathcal{A}$ is composed of two sub-tasks a_i and b_i (the same notations are used for the processing time: $a_i \in \mathbb{N}$ and $b_i \in \mathbb{N}$), and separated by a fixed idle time $L_i \in \mathbb{N}$ (see fig. 1(a)). For each i the second sub-task b_i must start its execution exactly L_i time units after the completion time of a_i .

A valid schedule $\sigma : \mathcal{A} \rightarrow \mathbb{N}$ consists in determining the starting time of each sub-task a_i of each task $A_i \in \mathcal{A}$. The tasks are processed on a single processor while preserving the constraints given by the compatibility graph.

The notation $\sigma(A_i)$ denotes the starting time of the task A_i . We use the following abuse of notation: $\sigma(a_i) = \sigma(A_i)$ (resp. $\sigma(b_i) = \sigma(A_i) + a_i + L_i$) denotes the starting time of the first sub-task a_i (resp. the second sub-task b_i). Let $C_{max} = \max_{A_i \in \mathcal{A}} (\sigma(A_i) + a_i + L_i + b_i)$ be the required time to complete all the tasks. Then

the objective is to find a feasible schedule which minimizes C_{max} . At last, using the notation scheme $\alpha|\beta|\gamma^1$ proposed by Graham and al. [10], the main problem denoted as Π will be defined by $1|coupled - task, (a_i, b_i, L_i), G_c|C_{max}$.

1.2 Related work

The problem of coupled-tasks has been studied in regard to different conditions on the values of a_i , b_i , L_i for $1 \leq i \leq n$, and precedence constraints [1,4,16]. Note that, in the previous works, all tasks are compatible by considering a complete graph [1,4,16]. Moreover, in presence of any compatibility graph, we find several complexity results [19,20], which are summarized² in Table 1:

Problem	Complexity
$1 coupled - task, (a_i = b_i = L_i), G_c C_{max}$	$\mathcal{NP}$ -complete
$1 coupled - task, (a_i = a, b_i = b, L_i = L), G_c C_{max}$	$\mathcal{NP}$ -complete
$1 coupled - task, (a_i = b_i = p, L_i = L), G_c C_{max}$	$\mathcal{NP}$ -complete
$1 coupled - task, (a_i = L_i = p, b_i), G_c C_{max}$	Polynomial
$1 coupled - task, (a_i, b_i = L_i = p), G_c C_{max}$	Polynomial

Table 1 Complexity for scheduling problems with coupled-tasks and compatibility constraints

Our work consists in measuring the impact of the compatibility graph on the complexity and approximation of scheduling problems with coupled-tasks on a mono processor. In this way, we focus our work on establishing the limits between polynomiality and $\mathcal{NP}$ -completeness of these problems according to some parameters, when the compatibility constraint is introduced. In [20,21], we have studied the impact of the parameter L , and have shown that the problem $1|coupled - task, (a_i = b_i = p, L_i = L), G_c|C_{max}$ was $\mathcal{NP}$ -complete as soon as $L \geq 2$, and polynomial otherwise. In the following, we complete complexity results with the study of other special cases according to the value of a_i and b_i , and we propose several approximation algorithms.

1.3 Organization of this paper

In the rest of the paper, we restrict our study to a special case, by adding new hypotheses to the processing time and idle time of the tasks. We consider the processing time a_i (resp. b_i) of all sub-tasks a_i (resp. b_i) equal to a constant a (resp. b), and $\forall i \in \{1, \dots, n\}$ the length of the idle time is L . Considering homogeneous tasks is a realistic hypothesis according to the tasks that the torpedo has to execute. Let $\Pi_1 : 1|coupled - task, (a_i = a, b_i = b, L_i = L), G_c|C_{max}$ be this new problem.

In section 2, we establish the complexity of Π_1 according to the values of a , b and L . On the one hand, we show that the problem is polynomial for any $L < a + b$. On the other hand, the problem becomes $\mathcal{NP}$ -complete for $L \geq a + b$. When $L = a + b$

¹ Where α denotes the environment processors, β the characteristics of the jobs and γ the criteria.

² The notation $a_i = a$ implies that for all $1 \leq i \leq n$, a_i is equal to a constant $a \in \mathbb{N}$. This notation can be extended to b_i and L_i with the constants b, L and $p \in \mathbb{N}$.

the problem may be considered as a new graph problem very close to the Minimum DISJOINT PATH COVER problem (Min-DPC). In section 3, we show that the problem is immediately 2-approximated by a simple approach, and we design a polynomial-time approximation algorithm with performance guarantee lower than $\frac{3}{2}$. In fact, we show that the approximation ratio obtained by this algorithm is between $\frac{3}{2}$ and $\frac{5}{4}$, according to the values of a and b . The last section is devoted to the study of Π_1 for some particular topology of the graph G_c . Some of presented results can be applied to the case $L > a + b$, which is not considered here and will be treated in future work.

2 Computational complexity

First, we prove that Π_1 is polynomial when $L < a + b$: it is obvious that a maximum matching in the graph G_c gives an optimal solution. Indeed, during the idle time L of a coupled-task A_i , we can process at most one sub-task a_j or b_k . Since the idle time L is identical, so it is obvious that finding an optimal solution consists in computing a maximum matching. Thus, the problem 1|coupled-task, ($a_i = a, b_i = b, L_i = L < a + b$), $G_c|C_{max}$ admits a polynomial-time algorithm with complexity $O(m\sqrt{n})$ (see [17]).

The rest of the paper is devoted to the case $L = a + b$. Without loss of generality, we consider the case³ of $b < a$. The particular case $b = a$ will be discussed in subsection 2.2.

2.1 From a scheduling problem to a graph problem

Let us consider a valid schedule σ of an instance $(\mathcal{A}, G_c)$ of Π_1 with $b < a$, composed of a set of coupled-tasks $\mathcal{A}$ and a compatibility graph G_c . For a given task A_i , at most two sub-tasks may be scheduled between the completion time of a_i and the starting time of b_i , and in this case the only available schedule consists in executing a sub-task b_j and a sub-task a_k during the idle time L_i with $i \neq j \neq k$ such that $\sigma(b_j) = \sigma(a_i) + a$ and $\sigma(a_k) = \sigma(a_i) + a + b$. Figure 2 shows a such configuration.

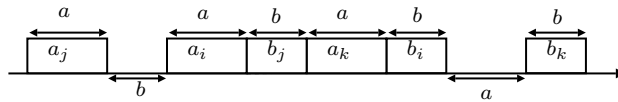


Fig. 2 At most two sub-tasks may be scheduled between a_i and b_i

We may conclude that any valid schedule σ can be viewed as a partition $\{T_1, T_2, \dots, T_k\}$ of $\mathcal{A}$, such that for any T_i the subgraph $P_i = G_c[T_i]$ of G_c induced by vertices T_i is a path (here, isolated vertices are considered as paths of length 0). Clearly, $\{P_1, P_2, \dots, P_k\}$ is a partition of G_c into vertex-disjoint paths. Figure 3 shows an instance of Π_1 (fig. 3(a)), a valid schedule (fig. 3(c)) - not necessarily an optimal one -, and the corresponding partition of G_c into vertex-disjoint paths (fig. 3(b)).

³ The results we present here can be symmetrically extended to the instances with $b > a$.

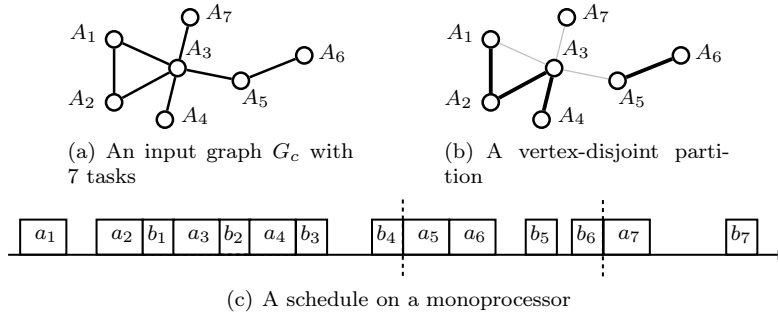


Fig. 3 Relation between a schedule and a partition into vertex-disjoint paths

For a given feasible schedule σ , let us analyse the relation between the length of the schedule C_{max} and the corresponding partition $\{P_1, P_2, \dots, P_k\}$ into disjoint-vertex paths. Clearly, we have $C_{max} = t_{seq} + t_{idle}$ where $t_{seq} = n(a + b)$ and t_{idle} is the inactivity time of the processor. Since t_{seq} is fixed for a given instance, t_{idle} obviously depends on the partition: for any path of length greater than 1, t_{idle} is incremented by $(a + b)$ (fig. 4(a)). A path of length 0 corresponds to a single task which increments t_{idle} by $L = a + b$. For any path of length 1, t_{idle} is only increased by a , because the two corresponding tasks may be imbricated as on figure 4(b).

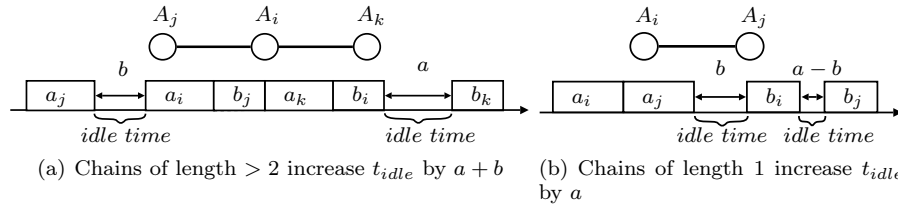


Fig. 4 Impact of the length of the paths on the idle time

Thus, finding an optimal schedule may be considered as a graph problem that we call **Minimum SCHEDULE-LINKED DISJOINT-PATH COVER (Min-SLDPC)** defined as follows:

Instance : a graph $G = (V, E)$ of order n , two natural integers a and b with $b < a$.

Result : a partition $\mathcal{P}$ of G into vertex-disjoint paths (which can be of size 0)

Objective : Minimize $n(a + b) + \sum_{p \in \mathcal{P}} w(p)$ where $w : \mathcal{P} \rightarrow \mathbb{N}$ is a cost function such that $w(p) = a$ if and only if $|E(p)| = 1$, and $w(p) = a + b$ otherwise.

In any solution, each path increments the cost of the idle time by at least a (when the path has a length 1), and at most $a + b < 2a$. So, we can deduce that an optimal solution to Min-SLDPC consists in finding a partition $\mathcal{P}$ with a particular cardinality k^* , and a maximal number of paths of length 1 among all possible k^* -partitions.

Clearly, Min-SLDPC is equivalent to Π_1 with $b < a$ and $L = a + b$, and can be viewed as the graph problem formulation of a scheduling problem. This problem is very close to the well-known **Minimum DISJOINT PATH COVER problem (Min-DPC)** which consists in covering the vertices of a graph with a minimum number of vertex-disjoint paths⁴. This problem has been studied in depth in several graph classes: it is known that this problem is polynomial on cographs [15], blocks graphs and bipartite permutation graphs [22], distance-hereditary graph [12], and on interval graphs [2]. In [5] and [9], the authors have proposed (independently) a polynomial-time algorithm in the case where the graph is a tree. Few years later, in [13] the authors showed that this algorithm can be implemented in linear time. Among the other results, there is a polynomial-time algorithm for the cacti [14], and another for the line graphs of a cactus [7]. In circular-arc graphs the authors [11] have proposed an approximation algorithm of complexity $O(n)$, which returns an optimal number of paths to a nearly constant additive equal to 1.

The problem Min-DPC is directly linked to the **HAMILTONIAN COMPLETION problem** denoted by **HC** [8], which consists in finding the minimum number of edges, noted $HC(G)$, that must be added to a given graph G , in order to make it hamiltonian (to guarantee the existence of a Hamiltonian cycle). It is known that if G is not hamiltonian, then the cardinality of a minimum disjoint path cover is clearly equal to $HC(G)$.

The dual of Min-DPC is found in the literature under the name Maximum DISJOINT-PATH COVER [8]. It consists in finding in G a collection of vertex-disjoint paths of length at least 1, which maximises the edges covered in G . This problem is known to be $\frac{7}{6}$ -approximable [3].

The following immediate theorem establishes the complexity of Min-SLDPC:

Theorem 1 *Min-SLDPC is an $\mathcal{NP}$ -hard problem.*

Proof Our proof is based on the polynomial-time transformation HAMILTONIAN PATH $\propto$ Min-SLDPC. Let us consider a graph G . The optimal solution of Min-SLDPC consists in finding a particular k -partition of G with k as small as possible. If k can be equal to 1, then the graph G contains an hamiltonian path. Reciprocally, if G contains an hamiltonian path, we can deduce a schedule with $C_{max} = (n + 1)(a + b)$: as $t_{seq} = n(a + b)$, t_{idle} must be equal to $(a + b)$, which is possible if and only if the schedule is represented with only one chain.

2.2 A particular case $\Pi_2:1|coupled - task, (a_i=b_i=p, L_i=L=2p), G_c|C_{max}$

Let us suppose, only in this subsection, that both sub-tasks are equal to a constant p and that the inactivity time is equal to a constant $L = 2p$.

The previous proof cannot be used for this case. Indeed the structure of these tasks allows to schedule three compatible tasks together without idle time (see figure 5). Another solution consists in covering vertices of G_c by triangles and paths (length 0 allowed), where we minimize the number of paths and then maximize the number of path of length 1.

This problem is a generalization of TRIANGLE PACKING [8] since an optimal solution without idle time consists in partitioning into triangles the vertices of G_c .

⁴ Sometimes referenced as the PATH-PARTITION problem (PP).

This problem is well known to be $\mathcal{NP}$ -complete, and leads to the $\mathcal{NP}$ -completeness of problem Π_2 .

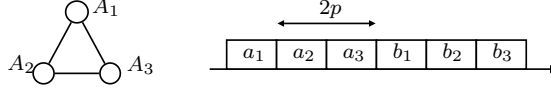


Fig. 5 Illustration of a schedule without idle time

A correct approximation algorithm for this problem is an algorithm close to the general case. Indeed, finding an optimal solution to this problem amounts to finding a covering of the graph G_c by triangles and paths, which minimize the idle time. In the following section, we will develop an efficient polynomial-time approximation algorithm for the general problem min-SLDPC.

3 Approximation algorithm for Min-SLDPC

First we show that any polynomial-time approximation algorithm⁵ admits a performance guarantee of 2. We also develop a polynomial-time $\frac{3}{2}$ -approximation algorithm based on a maximum matching in the graph G_c . In fact, we show that this algorithm has an approximation ratio of at most $\frac{3a+2b}{2a+2b}$, which leads to a ratio between $\frac{3}{2}$ and $\frac{5}{4}$ according to the values of a and b (with $b < a$). This result, which depends on the values a and b , will be discussed in section 4, in order to propose a better ratio on some class of graphs.

For any instance of Min-SLDPC, an optimal schedule has a length $C_{max}^{opt} = t_{seq} + t_{idle}^{opt}$ where $t_{seq} = n(a + b)$.

Remark 1 For any solution given by a heuristic h of cost C_{max}^h , we necessarily have⁶ $t_{idle}^h \geq (a + b)$ and also⁷ $t_{idle}^h \leq n(a + b)$. Then, for any solution h of Min-SLDPC we have a performance ratio $\rho(h)$ such that:

$$\rho(h) \leq \frac{C_{max}^h}{C_{max}^{opt}} \leq \frac{2n(a + b)}{(n + 1)(a + b)} < 2. \quad (1)$$

In the following, we develop a polynomial-time approximation algorithm based on a maximum matching in the graph G_c , with performance guaranty in $[\frac{5}{4}, \frac{3}{2}]$ according to the values of a and b .

Let I be an instance of our problem. An optimal solution is a disjoint-paths cover. The n vertices are partitioned in three disjoint sets: n_1 uncovered vertices, n_2 vertices

⁵ Between two independent tasks, idle time is not allowed when there are available tasks.

⁶ The equality is obtained when the graph G_c possesses an hamiltonian path, otherwise we need at least two paths to cover G_c (where G_c is not only an edge), which leads to increase t_{idle}^h by at least $2a \geq a + b$ units of time.

⁷ The worst case consists in executing tasks sequentially without scheduling any sub-task a_j or b_j of task A_j during the idle time of a task A_i .

covered by $\alpha_2 = \frac{n_2}{2}$ paths of length 1, and n_3 vertices covered by exactly α_3 paths of length strictly greater than 1 (see illustration figure 6). An optimal solution is equal to the sum of sequential time and idle time: $C_{max}^{opt} = n(a+b) + n_1(a+b) + \frac{n_2}{2}a + \alpha_3(a+b)$.

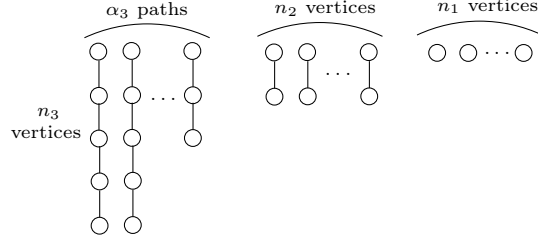


Fig. 6 Illustration of the optimal solution on an instance I

Now, we propose a polynomial-time approximation algorithm with non trivial ratio on an instance I . This algorithm is based on a maximum matching in G_c in order to process two coupled-tasks at a time. For two coupled-tasks A_i and A_j connected by an edge of the matching, we obtain an idle time of length a (see figure 4(b)).

With this algorithm, the n_1 uncovered vertices (resp. the n_2 vertices covered by edges) in an optimal solution are still uncovered (resp. covered by edges) with a maximum matching. At last, we consider the worst case in which the α_3 paths have odd lengths. So, from the n_3 vertices, there are α_3 vertices uncovered and $(n_3 - \alpha_3)$ vertices covered with a maximum matching. The upper bound is:

$$\begin{aligned} C_{max}^h &= n(a+b) + n_1(a+b) + \frac{n_2}{2}a + \alpha_3(a+b) + (n_3 - \alpha_3)\frac{a}{2} \\ &\leq \frac{3na}{2} + nb + \frac{\alpha_3 a}{2} + \alpha_3 b + \frac{n_1 a}{2} + n_1 b \end{aligned}$$

Obviously, the worst case is when $n_1 = n_2 = 0$, $\alpha_3 = 1$ and $n_3 = n$. Thus, we obtain for the relative performance:

$$\rho(h) \leq \frac{C_{max}^h}{C_{max}^{opt}} \leq \frac{n(a+b) + (a+b) + (n-1)\frac{a}{2}}{n(a+b) + (a+b)} \leq \frac{3a+2b}{2a+2b} \quad (2)$$

4 Instances with particular topologies

We conclude this work by a study of min-SLDPC when G_c admits a particular topology. We show that finding a ρ_{dpc} -approximation for Min-DPC on G_c allows to find a strategy with performance ratio $\rho_{sldpc} \leq \min\{\rho_{dpc} \times (\frac{a+b}{a}), \frac{3a+2b}{2a+2b}\}$. This leads to propose, independently from the values a and b , a $\frac{1+\sqrt{3}}{2}$ -approximation for min-SLDPC when Min-DPC can be polynomially solved on G_c .

From the literature, we know that Min-DPC is polynomial on trees [9, 13, 5], distance-hereditary graphs [12], bipartite permutation graphs [22], cactis [14] and many others classes. There are currently no result about the complexity of min-SLDPC on such

graphs: since the values of a and b have a high impact, techniques used to prove the polynomiality of Min-DPC cannot be adapted to prove the polynomiality of Min-SLDPC. Despite all our effort, the complexity of Min-SLDPC remains an open problem. However using known results on Min-DPC, we show how the approximation ratio can be decreased for these class of graphs. We propose the following lemma:

Lemma 1 *If min-DPC can be polynomially solved, then there exists a polynomial-time $\frac{(a+b)}{a}$ -approximation for min-SLDPC.*

Proof Let $I_1 = (G)$ be an instance of min-DPC, and $I_2 = (G, a, b)$ an instance of min-SLDPC. Let $\mathcal{P}_1^*$ be an optimal solution of min-DPC of cost $|\mathcal{P}_1^*|$, and $\mathcal{P}_2^*$ an optimal solution of min-SLDPC of cost OPT_{sldpc} . According to the definition of min-SLDPC, we have $|\mathcal{P}_2^*| \geq |\mathcal{P}_1^*|^8$. Since each path of a min-SLDPC solution increments the cost of the solution by at least a , then we have:

$$OPT_{sldpc} = \sum_{p \in \mathcal{P}_2^*} w(p) + n(a+b) \geq a|\mathcal{P}_2^*| + n(a+b) \geq a|\mathcal{P}_2^*| \quad (3)$$

$$\Rightarrow \frac{OPT_{sldpc}}{a} \geq |\mathcal{P}_2^*| \quad (4)$$

Let us analyse the cost given by the partition $\mathcal{P}_1^*$ if we consider it as a solution (not necessarily an optimal) to the instance I_2 of min-SLDPC. Since each path of a min-SLDPC solution increments the cost of the solution by at most $a+b$, then we have:

$$\begin{aligned} \sum_{p \in \mathcal{P}_1^*} w(p) + n(a+b) &\leq (a+b)|\mathcal{P}_1^*| + n(a+b) \leq (a+b)|\mathcal{P}_2^*| + n(a+b) \\ &\leq b|\mathcal{P}_2^*| + OPT_{sldpc} && \text{according to (3)} \\ &\leq \frac{b}{a}OPT_{sldpc} + OPT_{sldpc} && \text{according to (4)} \\ &\leq \frac{a+b}{a}OPT_{sldpc} \end{aligned} \quad (5)$$

The same proof may be applied if there exists a ρ_{dpc} -approximation for min-DPC, and then there exists a $\rho_{dpc} \times (\frac{a+b}{a})$ -approximation for min-SLDPC. Let us suppose that we know a constant ρ_{dpc} such that there exists a ρ_{dpc} -approximation for min-DPC on G_c . Let S_1 be the strategy, which consists in determining a $\rho_{dpc} \times (\frac{a+b}{a})$ -approximation for min-SLDPC from ρ_{dpc} , and S_2 the strategy, which consists in using the algorithm introduced in section 3. Clearly, S_1 is particularly relevant when b is very small in comparison with a . Whereas S_2 gives better ratio when b is close to a . Both strategies are complementary along the value of b , which varies from 0 to a . Choosing the best result between the execution of S_1 and S_2 , gives a performance ratio ρ_{sldpc} such that:

$$\rho_{sldpc} \leq \min \left\{ \rho_{dpc} \times \left(\frac{a+b}{a} \right), \frac{3a+2b}{2a+2b} \right\}. \quad (6)$$

Compared to executing S_1 only, this new strategy increases the obtained results if and only if ρ_{dpc} is lower than $\frac{3}{2}$ (see fig. 5).

⁸ The best solution for min-SLDPC is not necessarily a solution with a minimum cardinality of k .

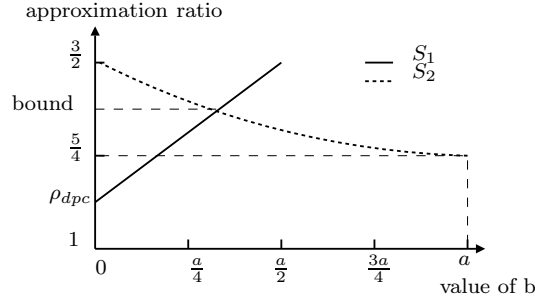


Fig. 7 Finding a good ρ_{dcp} -approximation helps to increase the results of section 3

We propose the following remark, which is not good news:

Remark 2 There is no ρ_{dpc} -approximation for min-DPC in general graphs for some $\rho_{dpc} < 2$

This result is a consequence of the impossibility theorem [6]. It can be checked by considering an instance of min-DPC which has an hamiltonian path: the optimal solution of Min-DPC has cost 1, thus any polynomial-time ρ_{dpc} -approximation algorithm with $\rho_{dpc} < 2$ will return a solution of cost 1, which is not allowed under the assumption that $\mathcal{P} \neq \mathcal{NP}$. This result also implies that constant-factor approximation algorithms for max-DPC do not necessarily give the same performance guarantees on min-DPC, since the best approximation ratio for max-DPC is $\frac{7}{6}$, which is lower than the inapproximability bound for min-DPC.

It is good news that Min-DPC is polynomial for some compatibility graphs such as trees [9,13,5], distance-hereditary graphs [12], bipartite permutation graphs [22], cactis [14] and many others classes; thus $\rho_{dpc} = 1$. For all these graphs we obtain an approximation ratio of $\min\{\frac{(a+b)}{a}, \frac{3a+2b}{2a+2b}\}$ which is maximal when $\frac{(a+b)}{a} = \frac{3a+2b}{2a+2b}$, i.e.:

$$\frac{(a+b)}{a} = \frac{3a+2b}{2a+2b} \Leftrightarrow -a^2 + 2ab + 2b^2 = 0. \quad (7)$$

The only solution of this equation with a and $b \geq 0$ is $a = b(1 + \sqrt{3})$. By replacing a by this new value on $\frac{(a+b)}{a}$ or on $\frac{3a+2b}{2a+2b}$, we show that in the worst case the approximation ratio is reduced from $\frac{3}{2}$ down to $\frac{1+\sqrt{3}}{2} \approx 1.37$.

5 Conclusion

We investigate a particular coupled-tasks scheduling problem Π_1 in presence of a compatibility graph. We have shown how this scheduling problem can be reduced to a graph problem. We have proved that adding the compatibility graph leads to the $\mathcal{NP}$ -completeness of Π_1 , whereas the problem is obviously polynomial when there is a complete compatibility graph (each task is compatible with each other). We have proposed a ρ -approximation of Π_1 where ρ is between $\frac{3}{2}$ and $\frac{5}{4}$ according to value of a and b . We have also decreased the upper bound of $\frac{3}{2}$ down to ≈ 1.37 on instances where

the Minimum DISJOINT PATH COVER problem can be polynomially solved on the compatibility graph.

The perspectives this work opens are measuring the pertinence of our polynomial-time approximation algorithm and classify $\Pi_1 : 1|coupled - task, (a_i = a, b_i = b, L_i = L)|C_{max}$ with complete compatibility graph.

References

1. D. Ahr, J. Békési, G. Galambos, M. Oswald, and G. Reinelt. An exact algorithm for scheduling identical coupled-tasks. *Mathematical Methods of Operations Research*, 59:193–203(11), June 2004.
2. S. Rao Arikati and C. Pandu Rangan. Linear algorithm for optimal path cover problem on interval graphs. *Information Processing Letters*, 35(3):149–153, 1990.
3. P. Berman and M. Karpinski. 8/7-approximation algorithm for (1,2)-tsp. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 641–648, New York, NY, USA, 2006. ACM.
4. J. Blazewicz, K.H. Ecker, T. Kis, and M. Tanas. A note on the complexity of scheduling coupled-tasks on a single processor. *Journal of the Brazilian Computer Society*, 7(3):23–26, 2001.
5. F. T. Boesch S. Chen and B. McHugh. On covering the points of a graph with point-disjoint paths. *Graphs and Combinatorics*, 406:201–212, 1974.
6. Ph. Chrétienne and C. Picouleau. Scheduling with communication delays: a survey. In *Scheduling theory and its applications*, pages 641–648. John Wiley & sons, 1995.
7. P. Detti and C. Meloni. A linear algorithm for the hamiltonian completion number of the line graph of a cactus. *Discrete Applied Mathematics*, 136(2-3):197–215, 2004.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-completeness*. Freeman, 1979.
9. S. E. Goodman, S. T. Hedetniemi, and P. J. Slater. Advances on the hamiltonian completion problem. *Journal of the ACM*, 22(3):352–360, 1975.
10. R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
11. R.-W. Hung and M.-S. Chang. Solving the path cover problem on circular-arc graphs by using an approximation algorithm. *Discrete Applied Mathematics*, 154(1):76–105, 2006.
12. R.-W. Hung and M.-S. Chang. Finding a minimum path cover of a distance-hereditary graph in polynomial time. *Discrete Applied Mathematics*, 155(17):2242–2256, 2007.
13. S. Kundu. A linear algorithm for the hamiltonian completion number of a tree. *Information Processing Letters*, 5:55–57, 1976.
14. S. Moran and Y. Wolfstahl. Optimal covering of cacti by vertex-disjoint paths. *Theoretical Computer Science*, 84:179–197, 1988.
15. K. Nakano, S. Olariu, and A. Y. Zomaya. A time-optimal solution for the path cover problem on cographs. *Theoretical Computer Science*, 290(3):1541–1556, 2003.
16. A.J. Orman and C.N. Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72:141–154, 1997.
17. A. Schrijver. *Combinatorial Optimization : Polyhedra and Efficiency (Algorithms and Combinatorics)*. Springer, July 2004.
18. R.D. Shapiro. Scheduling coupled tasks. *Naval Research Logistics Quarterly*, 27:477–481, 1980.
19. G. Simonin. Étude de la complexité de problèmes d’ordonnancement avec tâches-couplées sur monoprocesseur. *Majestic '08*, 2008.
20. G. Simonin, R. Giroudeau, and J.-C. König. Complexity and approximation for scheduling problem for a torpedo. 2009. To appear in CIE'39, Troyes, France.
21. G. Simonin, R. Giroudeau, and J.-C. König. Extended matching problem for a coupled-tasks scheduling problem. 2009. To appear in TMFCS, Orlando, Florida.
22. R. Srikant, R. Sundaram, K. Sher Singh, and C. Pandu Rangan. Optimal path cover problem on block graphs and bipartite permutation graphs. *Theoretical Computer Science*, 115(2):351–357, 1993.

Scheduling jobs with release times preemptively on a single machine to minimize the number of late jobs

Nodari Vakhania

Abstract We give a direct combinatorial $O(n^3 \log n)$ algorithm for minimizing the number of late jobs on a single machine when jobs have release times and preemptions are allowed. Our algorithm improves the earlier $O(n^5)$ and $O(n^4)$ dynamic programming algorithms for this problem.

1 Introduction

We are asked to assign or schedule n jobs $\mathcal{J} = \{1, 2, \dots, n\}$ on a single machine. Each job $j \in \mathcal{J}$ has the *processing time* p_j (time that it needs on the machine), the release time r_j (time moment when it becomes available) and due date d_j (its desired completion time). These are integer numbers. A job might be *preempted*, i.e., split into portions each portion being assigned at a different time interval to the machine. A *schedule* assigns each job j to the machine at the specified time moment(s) no less than r_j with the total duration of p_j so that no two jobs are assigned to the machine at any time moment (i.e., job execution intervals cannot overlap in time). A job is *late* (*on time*, respectively) if it is completed after (at or before, respectively) its due date. In our model it is important to complete as many jobs as possible by their due dates which is our objective, i.e., we aim to minimize the number of late jobs. Without loss of generality, we assume that job release times and due dates are *consecutive*, that is, $r_j + p_j \leq d_j$, for each j (every job may potentially be completed by its due date).

Our problem abbreviated $1/r_j; pmtn / \sum U_j$ (here U_j is a 0-1 function taking value 1 iff job j is late) is known to be polynomial (though its non-preemptive version is strongly NP-hard). The first polynomial dynamic programming algorithm with the time complexity of $O(n^5)$ was developed by Lawler [2]. The result was improved by Baptiste [1] who has suggested another dynamic programming algorithm with the time complexity $O(n^4)$ (Baptiste also overviews some related results). Here we further improve the time complexity proposing a direct combinatorial $O(n^3 \log n)$ algorithm. The optimal schedule is generated by the Earliest Due-date heuristic (ED-heuristic). Iteratively, at each scheduling time with a partial ED-schedule, we determine which of

Science faculty, State University Morelos, Mexico.
Partially supported by CONACyT grant 48433.
E-mail: nodari@uaem.mx

the currently omitted jobs are to be included at the expense of the omission of some currently scheduled job. The number of iterations is bounded by $O(n)$, whereas our decision at each iteration has the cost $O(n^2 \log n)$.

2 Some basics

The late jobs (if any) can be arbitrarily be appended to a schedule that contains maximal possible number of on-time scheduled jobs. In this way we easily obtain an optimal schedule from the former schedule. From now on we shall concentrate our efforts on finding such a schedule which we shall also call optimal. In general, there will be no late job in any schedule we shall be dealing with. We denote by $|S|$ the number of jobs (completed on time) in a schedule S , and by $||S||$ the completion time of the latest completed job (the makespan) in S . We will use S for both, a schedule and the corresponding set of jobs.

We apply (preemptive) ED-heuristic for scheduling the jobs: iteratively, among all released jobs by the current scheduling time t , ED-heuristic schedules a job with the smallest due-date ties being broken by selecting a shortest job (further ties are broken arbitrarily). If during the execution of a job another job with a smaller due-date or with the same due-date but with a smaller early completion time is released (i.e., this job, if immediately scheduled, will be completed earlier than the current job), the former job is interrupted and the latter job is scheduled. We call the job selected by ED-heuristic at the current scheduling time t the *incoming job* and denote it by $i(t)$. Initially, $t = \min\{r_j | j \in J\}$. Each new value of t is either the release time of the next incoming job or/and the completion time of the latest scheduled job so far. In the former case, if the machine is not idle, the currently running job j is interrupted by the next incoming job i with $d_i < d_j$, or with $d_i = d_j$ but with a smaller early completion time. We denote by t^- (t^+ , respectively) the previous (the next, respectively) to t scheduling time, and by S^t the schedule constructed to time t . In general, we write $t \succ t'$ if t is a scheduling time occurred after scheduling time t' in our algorithm ($t^+ < t$ is possible, in general, we may have $t < t'$ for $t \succ t'$).

At any scheduling time t the set of the already scheduled jobs can be divided into the ones which are completed by time t and the ones which were interrupted and yet not completed by time t . We denote the latter set of jobs by U_t , and denote by u^+ the *residue* of $u \in U_t$; that is, the part of u yet unprocessed by time t . Note that a new uncompleted job arises only when the incoming job interrupts it. For technical reasons, it will be convenient to “complete virtually” jobs in U_t after time t . We shall create respectively the *virtual part* in S^t that starts after time t . We shall refer to the part of S^t before time t as its *real part*. From now on, when referring to an ED-schedule S we will mean its both real and virtual parts; however, the makespan $||S||$ will concern only the real part of S . Below we describe how we schedule residues without any interruption in the virtual part.

Recall that if $i(t)$ is more urgent than the currently running job j then $i(t)$ interrupts j , and j^+ is non-empty. In general, we schedule the residues of jobs in U_t applying a dislodged ED-heuristic. Since all jobs up to time t are scheduled by ED-heuristic, each $i(t)$ has a due-date, no greater than that of any job from U_t . This implies that up to time t , the jobs from U_t are scheduled by the non-increasing order of their due dates. Each newly included job in U_t has a due date no greater than that of all earlier included ones. Hence, ED-heuristic will schedule residues of jobs from U_t in the order, reverse

to the order in which the completed parts before time t were scheduled. We conserve this order for scheduling residues after time t , but we schedule them as late as possible, without making them late, as follows. The earliest arisen uncompleted job is scheduled so that it is completed exactly at its due date. Each newly arisen uncompleted job u has a due date, no greater than that of all earlier arisen uncompleted jobs with already “scheduled” residues. Let $\tau > t$ be the starting time of the latest scheduled residue so far. If $d_u \geq \tau$ then we add u^+ so that it completes at time τ ; otherwise, we add u^+ so that it completes at time d_u . Observe that no overlapping between residues may occur and that each residue will start after time t in S^t . However, some residue may start before time $t + p_i$, $i = i(t)$.

Suppose among all (yet uncompleted) jobs released by time t , job $u \in U_t$ has the smallest due date. Then u is the lastly arisen uncompleted job and u^+ is the earliest started residue in S^t . We schedule job u at time t and update the virtual part for S^{t^+} by removing the already scheduled part of u^+ . Note that since u^+ was the earliest scheduled residue, no new gap in between the remained residues will occur. If u is no more interrupted and hence completely scheduled, we let $U_{t^+} := U_t \setminus \{u\}$. Otherwise, u is interrupted once again, u^+ is updated respectively t^+ being the release time of the next incoming job.

Suppose job j completes at time t and $U_t = \emptyset$. Then we will say that the next incoming job (released no earlier than at time t) starts new *block*. Each S^t is naturally divided into the blocks: at scheduling time t^+ a new block is initiated if $[t, t^+)$ is a gap (we assume to have a zero-length gap at time t between two blocks if the next incoming job is released right at time t). By scheduling time t , we will denote the current block by B^t . If B^t is not a new block then either $i(t)$ interrupts the currently running job or a job from U_t is resumed.

Let $S = S^t$. We denote by $S(-k)$, $k \in S$ ($S(+l)$, $l \notin S$, respectively) the ED-schedule generated for the job set $S \setminus \{k\}$ ($S \cup \{l\}$, respectively) the virtual part being generated the dislodged ED-heuristic as above. We may generalize these notations straightforwardly substituting k and l by job sets.

We will say that the incoming job i can be *perfectly scheduled* at time t if $S^t(+i)$ is a feasible schedule in which all jobs are completed on time. We may observe that i can be perfectly scheduled only if the latest arisen residue starts at or after time $t + p_i$ in $S^t(+i)$ (as otherwise the overlapping will occur). Let us call a job set *feasible* if there exists a feasible schedule in which all the jobs from this set are scheduled on time. The following observation is easily verified:

Observation 1 *If $i(t)$ cannot be perfectly scheduled then the job set $S \cup \{i\}$ is not feasible.*

It follows that if $i = i(t)$ cannot be perfectly scheduled then either i is late in $S^t(+i)$ or/and it overlaps with the virtual part. Iteratively at each scheduling time t we start job $i(t)$ at time t if it can be perfectly scheduled. The real part of the resultant schedule $S^{t^+} = S^t(+i)$ now extends up to time t^+ . It is less straightforward the case when i cannot be perfectly scheduled. We introduce some extra definitions first.

We will say that the incoming job i is *disregarded* (*omitted*, respectively) if it is discarded at time t and is never again considered for the inclusion (discarded at time t but can later again be considered for the inclusion, respectively). An omitted job at time t is placed into our current *reserve list* L_t .

We say that S^t *cannot be enlarged* if there exists no feasible set with $|S^t| + 1$ jobs from $S^t \cup L_t \cup \{i(t)\}$. This means that $i(t)$ cannot be scheduled on time even by

removing any subset of already scheduled jobs from S^t and including at least the same amount of jobs from L_t instead.

σ , a subset of jobs in S^t , is said to give an *admissible choice* at time t if the job set $S^t \setminus \sigma \cup \{i\}$ is feasible. When σ is a single-element set, a job with an admissible choice will be referred to as an *admissible job*. We shall refer to the earliest scheduling time t on which $k \in S^t$ gives an admissible choice as the *check-in time* of k , denoted by $\tau(k)$.

2.1 Pending gaps and filling jobs

If the incoming job $i(t)$ cannot be perfectly scheduled then either $i(t)$ itself or some already scheduled job from S^t is to be omitted (Observation 1). It is easy to see that an appropriate candidate for the omission is to be an admissible job $j \in S^t$. We shall refer to the set of all time intervals occupied by j in S^t as the *(potential) liberated space* by j denoted by $\mathcal{I}_j$. If we omit job j then the intervals from $\mathcal{I}_j$ will be used by other jobs of S^t and/or the incoming job(s). In $S^t(-j)$, some intervals from $\mathcal{I}_j$ or their parts may remain idle, and because of a possible left-shift of some jobs from S^t , there may also occur gaps after the intervals from $\mathcal{I}_j$. We call either of these idle intervals which may potentially be arisen (in case j is omitted) *(potential) pending gaps* of j at time t and denote the corresponding set by $pg_t(j)$.

Recall that if $i = i(t)$ cannot be perfectly scheduled then either (1) i is late or (2) i is not late but it overlaps with the virtual part of S or (3) i is late and it also overlaps with the virtual part of S . Assume i is late (cases (1) and (3)). As job release times and due dates are consecutive, i must have been released before time t , hence it is delayed by some already scheduled job(s). There is at least one $j \in S^t$ at least as urgent as i and completed by time t (in particular, the latest scheduled job in S^t has this property). The *left-shift* by $j \in S^t$ in favor of job i is the difference $\|S^t(+i)\| - \|S^t(-j+i)\|$, and $\|S^t\| - \|S^t(-j)\| \leq p_j$ is the left-shift just by job j (in these formulas only the corresponding real parts are considered).

Let k be the latest scheduled job (job portion) in B^t with $d_k \geq d_i$. Let us call any admissible job scheduled after job k in B^t *category I* job (if there is no k , all admissible jobs from B^t are category I jobs). The left-shift yielded by each category I job j in favor of i is at least $\min\{p_i, p_j\}$ if j is started no earlier than at time r_i . Job i cannot be immediately scheduled within the liberated space by j if j is completed by time r_i . In this case, the jobs which were scheduled after j might be successively left-shifted (by ED-rule, all these jobs will be less urgent than j). Since a category I job is more urgent than i , it can have no residue at time t . If we omit the real part of any job from B^t scheduled before all category I jobs, then neither job i nor any category I job can be left-shifted. Hence, there may exist an admissible job from B^t which is not a category I job only if i overlaps with the virtual part of S^t in $S^t(+i)$ (case (2)). We call this intersection the *conflict interval* and will refer to the corresponding admissible jobs in B^t (scheduled before all category I jobs) as the *category II* jobs.

So there may occur a left-shift in $S^\tau(-j+i(\tau))$ only if j is a category I job; if there remains a space unused by the left-shifted jobs, some residue(s) may also be scheduled within that space. If there still remains an unused space, the pending gaps after the intervals from $\mathcal{I}_j$ will arise. A pending gap of j might either be from the real part of S^τ (a *real pending gap* of j) or it might also be from its virtual part (a *virtual pending gap* of j). A virtual pending gap of j will arise only if ED-heuristic has “moved” a residue to the real part in $S^\tau(-j)$.

The set $pg_t(j)$ will be altered if some its interval(s) can potentially be occupied by the next incoming job $i = i(t)$ that can be perfectly scheduled (and hence is included). In $S^t(-j + i)$, job i would be left-shifted (compared to its location in $S^t(+i)$) and the remained idle time intervals (released by i and/or possibly some other successively left-shifted jobs) will become new pending gap(s) of j . Observe that the total length of the pending gaps of j will not be altered.

Suppose now i cannot be perfectly scheduled and hence is omitted, whereas j gives an admissible choice at time t . We call any such an omitted job a (*potential*) *filling job* of j , and refer to jobs with a common filling job as *correlated*. We denote by $F_t(j)$ the set of filling jobs of j by time t . We shall refer to j as a *pending job* at time t if $F_t(j) \neq \emptyset$. A filling job of j may potentially be included into a later generated ED-schedule at the expense of the omission of job j . Meanwhile, such a job does not affect the current $pg_t(j)$. Note that in all above definitions j is an admissible job that still remains scheduled (we just consider a possibility of its omission).

3 Swaps and substitutions

Consider the following extension of ED-heuristic, EED-Algorithm.1 for short: At each scheduling time t schedule $i(t)$ till the next scheduling time t^+ if $i(t)$ can be perfectly scheduled; otherwise, disregard $i(t)$. Suppose $\iota = i(\tau)$ is the first disregarded job. Due to Observation 1, EED-Algorithm.1 remains optimal if each following incoming job is perfectly scheduled (EED-Algorithm.1 is clearly optimal if each incoming job is perfectly scheduled). Suppose another incoming job $i = i(t)$, $t \succ \tau$, cannot be perfectly scheduled. Clearly, S^t cannot be enlarged if there is no $k \in S^t$ with the feasible $S^t \setminus \{k\} \cup \{\iota, i\}$. Otherwise, ι and i must have been filling jobs of k , and k must have been given an admissible choice at times τ and t . So it is apparent that EED-Algorithm.1 is optimal unless there occurs a scheduling time t as indicated. We shall incorporate a corresponding substitution into EED-Algorithm.1 (note that so far, each k remained scheduled).

Let $D \subseteq F_t(k)$. The (*normal*) *swap* $k \leftrightarrow D$ accomplishes the substitution of job k with jobs from D (using the gaps from $pg_t(k)$): it omits k and schedules jobs in $D \cup S \setminus \{k\}$ by ED-heuristic, also updating the current reserve list: $S^{t+} := S^t(-k + D)$, $L^{t+} := L_t \setminus D \cup \{k\}$. The swap $k \leftrightarrow D$ (the set D) is *feasible* if $S^t(-k + D)$ is feasible. We shall only deal with feasible swaps. Observe that any feasible swap yields the same or higher number of on time scheduled jobs. $k \leftrightarrow D$ is *increscent* whenever $|D| \geq 2$.

We organize our algorithm so that it accomplishes an increscent feasible swap as soon as such a possibility occurs; i.e., as soon as a feasible $D \subseteq F_t(k)$ with $|D| = 2$ can be formed, for some pending job k . Since job release times and due-dates are consecutive, if the next $i = i(t)$ cannot be perfectly scheduled then it must be a filling job of at least one $k \in B^t$ with an admissible choice. If for all such k s i is the only filling job then it can clearly be omitted at time t (moved to the reserve list L_t). Suppose $|F_t(k)|$ just becomes 2 being feasible. If there are several such pending jobs let k be the latest scheduled one called the *substitution job* of time t . Then we carry out the swap $k \leftrightarrow F_t(k)$ at time t^+ . Then $|Fd_{t+}(j)|$ becomes no more than 1, for any admissible j .

We shall be also dealing with a non-increscent swap of the form $k \leftrightarrow i(t)$. Suppose among all pending jobs $j \in S^t$ with $|F_t(j)| > 1$ there is no j with feasible $F_t(j)$ (hence no increscent swap can be carried out), but there is $k \in S^t$ with the property that job i is completely “absorbed” within the space that might be released by k , that is,

$\|S^t(-k+i)\| < \|S^t\|$. If k has this property then it is called an *absorbent* job at time t . Among all absorbent jobs at time t we also break ties selecting the latest scheduled one, say k , and carry out the swap $k \leftrightarrow i$.

We have seen in the previous section that the set of pending gaps of a pending job $j \in S^{t+}$ can be altered when the new incoming job is scheduled. It can also be altered if the substitution of job k with jobs from D is carried out at time t , as new gaps after that substitution may arise (the space released by k may exceed the space used by jobs in D). We shall update $pg_{t+}(j)$ in either of these cases by actually generating a partial ED-schedule for the corresponding left-shifted jobs. For the technical convenience, we also include the above newly arisen gaps into $pg_{t+}(j)$ (so, $pg_{t+}(j)$ will contain every available interval for scheduling of the filling jobs of j in case it is omitted). Since there are less than n pending jobs, the total cost of the update of pending gaps of all pending jobs at time t is $O(n^2 \log n)$.

If none of the above two substitutions can be carried out at time t then $i(t)$ is omitted the set of filling jobs of each corresponding pending job being completed by job i . So, there will arise pending jobs k with $|F_t(k)| > 1$ and such that no two jobs in $F_t(k)$ form a feasible group. Assume $|F_t(k)|$ just becomes 2 (being non-feasible). Then $F_t(k) = \{i(\tau), i(t)\}$, where τ is the check-in time of k . By ED-heuristic, if among jobs scheduled in S^t after time τ there is one which is more urgent than $i(\tau)$ then it must have been released after time τ , and also must be shorter than $i(\tau)$. At each successive scheduling time t' , the current $F_{t'}(k)$ is to be complemented by $i(t')$ if it cannot be perfectly scheduled being a filling job of k . In general, this job may or may not form a feasible set with a job from $F_{t'}(k)$: let us call $F_t(k)$ a *conflict set* at time t if for every $l, m \in F_t(k)$ the set $S^t \setminus \{k\} \cup \{l, m\}$ is not feasible (we will also say that these jobs conflict). We have too many conflict jobs to be scheduled. We may discard some of them using the following dominance relation. Let l and j be conflict jobs of k with $r_l \leq r_j$, and let c_j be the completion time of j in $S^t(-k+j)$. Then j *dominates* l , written $j \text{ dom } l$, if $c_j \leq c_l$. It is easily verified that this is a transitive relation and that the following lemma holds:

Lemma 1 *If $j \text{ dom } l$ then l can be discarded. In particular, if any of the future incoming jobs will conflict with j then it will also conflict with l .*

Thus we wait until the earliest time moment t such that $i = i(t)$ forms a feasible set with one of the jobs from $F_t(k)$. We can of course verify this condition for each $j \in F_t(k)$ by forming the corresponding ED-schedule $S^t(-k+j+i)$. But this can be done more efficiently using the pending gaps of k . Let for $j \in F_t(k)$, λ_j be the total length of gaps in $pg_t(k)$ after time moment r_j . Since j is more urgent than i it will occupy all these gaps in $S^t(-k+j+i)$ unless their total length is more than p_j ; in the latter case j will completely be scheduled within the gaps in $pg_t(k)$. Let π_j be the portion of j that did not fit within the gaps in $pg_t(k)$. Further, let $\theta_i(j)$ be the total gap length in $pg_t(k)$ between time moments r_i and r_j if $r_i < r_j$, otherwise let $\theta_i(j)$ be 0. Then the portion of job i of the length $\min\{\theta_i(j), p_i\}$ will be scheduled within the gaps in $pg_t(k)$ before job j in $S^t(-k+j+i)$; let $\pi_i(j)$ be the rest of job i . It follows that job i will form a feasible set with $j \in F_t(k)$ with the minimal $\pi_j + \pi_i(j)$ if there exists such a job in $F_t(k)$. We can easily check this condition for each j in a constant time. Hence the cost of the verification/forming a feasible dependent set for a particular pending job k is $O(n)$. We first verify the condition for the latest scheduled pending job, in case of failure we try the next latest scheduled pending job, and so on. Hence the total cost of the verification/forming a feasible dependent set at time t is $O(n^2)$. If no feasible

set could have been formed then the set of filling jobs of each corresponding pending job is completed by job i which yields the total cost of $O(n)$.

4 Schedule alterations

The following notion is used to represent every possible job substitution: A *schedule alteration* (an *alteration* for short) $\mathcal{A}_t(o \rightleftharpoons \iota)$, $o, \iota \in \mathcal{N}^+$, yields a job rearrangement that omits some o jobs scheduled in S^t and includes ι jobs from L_t so that in the resultant schedule no job is late (we may omit some parameters in $\mathcal{A}_t(o \rightleftharpoons \iota)$ when this will cause no confusion). Technically, this schedule is obtained by the application of ED-heuristic to the altered job set. We may easily observe that feasible swap $k \leftrightarrow D$ is a schedule alteration of the form $\mathcal{A}(1 \rightleftharpoons |D|)$.

Given some initial scheduling time t , let us denote by $\{\mathcal{A}\}$ a sequence of schedule alterations the first of which is to be applied at time t (to the job set S^t), the second alteration is to be applied at time t^+ (to the job set S^{t^+}), and so on, each next alteration from $\{\mathcal{A}\}$ is to be applied at the currently derived scheduling time to the current job set.

A schedule alteration $\mathcal{A}_t(o \rightleftharpoons \iota)$ is *non-descending* (*increscent*, respectively) if $\iota \geq o$ ($\iota > o$, respectively). The *growth* of a non-descending schedule alteration is $growth[\mathcal{A}(o \rightleftharpoons \iota)] = \iota - o$. We define similarly the *overall growth*, $growth[\{\mathcal{A}\}]$ for the sequence $\{\mathcal{A}\}$ summing up all terms $\iota - o$ for each alteration in $\{\mathcal{A}\}$. We call a sequences of schedule alterations $\{\mathcal{A}\}$ and $\{\mathcal{A}'\}$ *equivalent* if they result in the same job rearrangement; i.e., the resultant sets of omitted and included jobs and hence the resultant schedules are the same (in particular, $growth[\{\mathcal{A}\}] = growth[\{\mathcal{A}'\}]$). The next lemma straightforwardly follows:

Lemma 2 *With each sequence of schedule alterations a unique equivalent schedule alteration is associated. Moreover, there is an increscent sequence of schedule alterations yielding an optimal schedule.*

We call a non-descending alteration *atomic* if there exists no equivalent sequence of 2 or more non-descending schedule alterations. Clearly, alterations of the form $1 \rightleftharpoons 1$ and $1 \rightleftharpoons 2$ are atomic. No other type of an atomic schedule alteration exists:

Lemma 3 *A non-descending schedule alteration that is not of the form $1 \rightleftharpoons 1$ or $1 \rightleftharpoons 2$ is not atomic. Furthermore, any increscent sequence of schedule alterations can be decomposed into an equivalent sequence of atomic schedule alterations of the form $1 \rightleftharpoons 1$ and $1 \rightleftharpoons 2$.*

Proof. We illustrate the proof idea for an alteration of the form $2 \rightleftharpoons 3$. Let k_1 and k_2 be the jobs to be omitted, and l_1, l_2 and l_3 be the jobs to be included. We have to include one more job than we omit. Then, as it is easily seen, one of the swaps $k_\iota \leftrightarrow l_\kappa$ must be feasible, $1 \leq \iota \leq 2$, $1 \leq \kappa \leq 3$. Hence, we have one schedule alteration of the form $1 \rightleftharpoons 1$. Without loss of generality, suppose it corresponds to the swap $k_1 \leftrightarrow l_1$. Then clearly, the swap $k_2 \leftrightarrow \{l_2, l_3\}$ must be another feasible swap. Hence, the initial alteration of the form $2 \rightleftharpoons 3$ was decomposed into two atomic alterations of the form $1 \rightleftharpoons 1$ and $1 \rightleftharpoons 2$. We can apply a similar construction to any non-descending schedule alteration. $\square$

Our next extension of ED-heuristic, EED-Algorithm_2, incorporates non-descending schedule alterations of the form $(1 \rightleftharpoons 2)$ and $(1 \rightleftharpoons 1)$ (an absorbent schedule alteration) into EED-Algorithm_1: At the next scheduling time t , schedule $i(t)$ if it can be perfectly scheduled. Otherwise, if there exists a pending job k such that $i = i(t)$ forms a feasible set $D^t(k) = \{i, j\}$, for some $j \in F_t(k)$ then carry out the swap $k \leftrightarrow D^t(k)$ (breaking ties by selecting the latest scheduled k and j yielding the minimal $\|S^t(-k + D^t(k))\|$) (we call $(k, D^t(k))$ the substitution pair of time t). Else, if there exists an absorbent job, substitute the latest scheduled absorbent job with $i(t)$, i.e., carry out the swap $k \leftrightarrow i(t)$. Else (none of the above can be done), omit $i(t)$. Based on the above two lemmas, we can say that there exists an optimal ED-rule based algorithm with atomic schedule alterations. We will soon come to such an algorithm by completing EED-Algorithm_2.

Lemmas 2 and 3 motivate, in the straightforward way, the following definition. An (atomic) alteration (swap) $\mathcal{A}_t$ omitting job k *remains sound* (by time τ , $\tau \succeq t$) if there exists no increscent schedule alteration including job k (at the scheduling times up to time τ). Whether $\mathcal{A}_t$ is sound at time τ depends on the release times of the incoming jobs between times t and τ (for example, $\mathcal{A}_t$ clearly remains sound if all above incoming jobs are released after time $\|S^t(-k + D^t(k))\|$).

Lemma 4 *EED-Algorithm_2 is optimal if every accomplished (atomic) alteration remains sound (by the latest occurred scheduling time).*

Proof. The proof is more or less straightforward. We give a sketch. By Lemma 2 it will suffice to show that there may exist no increscent schedule alteration. By Lemma 3, any such an alteration can be decomposed into a sequence of atomic schedule alterations. Let us now see what kind of atomic schedule alterations including any omitted job may exist. First recall that every omitted job is either an omitted incoming job or a substitution job. The latter case needs no special treatment here as every schedule alteration remains sound (so there may exist no increscent schedule alteration including any omitted substitution job). In the former case, we know that any omitted incoming job j is a filling job of at least one currently scheduled job. Since j was not included, there existed no corresponding absorbent alteration. A non-absorbent alteration of the form $1 \rightleftharpoons 1$ may only make late some currently scheduled job(s) and cannot be of any benefit. Now, an alteration of the form $1 \rightleftharpoons 2$ including job j is either not feasible or will force the omission of some alternative filling job j' have been included by EED-Algorithm_2. The former case is obvious, whereas recall that in the later case ties should have been broken by selecting j' yielding the minimal makespan and hence the minimal right-shift for the later scheduled jobs. Similarly as above, by including job j and omitting j' we may only make some currently scheduled job late and this cannot give us any benefit. Thus there are no atomic alterations including any omitted jobs that may result in an increscent schedule alteration, which proves the lemma. $\square$

Due to the above lemma, we will complete our algorithm by specifying how to deal with schedule alterations that do not remain sound. We will say that $k \in L_t$ can be *freely returned* at time t if $S^t \cup \{k\}$ is feasible.

Lemma 5 *If the swap $k \leftrightarrow D^t(k)$ does not remain sound by time $t' \succ t$ then job k can be freely returned at time t' .*

Proof. The proof is quite simple and straightforward. We give some details. We need to verify whether there arises an increscent schedule alteration re-including job k at time t' . Such an alteration cannot exist if all jobs from $D = D^t(k)$ remain in $S^{t'}$ and

no job correlated with k was omitted from time t to time t' . Suppose first no such a correlated job was omitted while some job(s) from D have been omitted by time t' . Such a job j could have been omitted only if it was selected as a substitution job; some filling jobs of j should have been included instead. If after the re-inclusion of k one of these jobs become late then the corresponding swap cannot be increment and hence $k \leftrightarrow D^t(k)$ remains sound. Otherwise, k can be freely returned. Now, if some job l correlated with k was omitted then some job(s) from D could have been potentially rescheduled within the pending gaps of l . Similarly as above, if after the re-inclusion of k a job scheduled within the pending gaps of l become late then the swap $k \leftrightarrow D^t(k)$ remains sound. Otherwise, k can be freely returned. $\square$

Below we give a formal description of our final algorithm which complements the EED-Algorithm.2 by verifying if any of the omitted substitution jobs can be freely returned (by our convention, $S^{t^+} := S^t(+i)$ yields scheduling of job $i = i(t)$ from time t up to the next scheduling time t^+):

EED-Algorithm.3

initial settings: $t^+ := \min\{r_j | j \in J\}$; $S^{t^+} := \emptyset$; $L^{t^+} := \emptyset$;

BEGIN PROCEDURE MAIN

IF there exists $i(t^+)$ THEN $t := t^+$; **PROCESS**($i(t)$) ELSE STOP

END PROCEDURE MAIN

BEGIN PROCEDURE PROCESS(i)

IF i can be perfectly scheduled THEN $S^{t^+} := S^t(+i)$

ELSE

IF there exists an admissible k with feasible $D^t(k)$

$\{D^t(k) = \{i, l\}, \text{ for some } l \in F_t(k)\}$

THEN $S^{t^+} := S^t(-k + D^t(k))$;

{breaking ties by selecting the latest scheduled k }

FOR each substitution job j that can be freely returned

{include that job} $S^{t^+} := S^{t^+}(+j)$

ELSE {there is no feasible set with two jobs}

IF there is an absorbent job

THEN select the latest scheduled one k ; $S^{t^+} := S^t(-k + i)$

ELSE {omit i } $L^{t^+} := L_t \cup \{i\}$

END PROCEDURE PROCESS(i).

Theorem 1 *EED-Algorithm.3 finds an optimal schedule in time $O(n^3 \log n)$.*

Proof. The soundness part immediately follows from Lemmas 4 and 5. As to the time complexity, first we note that the total number of scheduling times is $O(n)$. Indeed, at any scheduling time an incoming job is either a new (yet unconsidered) job or it is a job from the current virtual part. The total number of scheduling times corresponding to the first case equals to the number of the new incoming jobs, i.e., it is n . In the second case, a job from the current virtual part may become an incoming job for the second, third or more times. But such a job, if not completely scheduled, might be interrupted only by a new arrived incoming job. Then the total number of times on which all jobs from the virtual part may become an incoming job is $O(n)$. Hence, the total number of scheduling times is $O(n)$.

If all incoming jobs can be perfectly scheduled then EED-Algorithm_3 behaves as ED-heuristic with the time complexity $O(n \log n)$. Suppose the incoming job $i = i(t)$ cannot be perfectly scheduled. We first verify whether a substitution might be accomplished at time t . For that, we check whether there are pending jobs in S^t for which $i(t)$ becomes a filling job. We try such pending jobs in the decreasing order of their ordinal numbers in S^t starting with the latest scheduled job in S^t . As we saw at the end of Section 3, the cost of the verification/forming of a feasible set at time t is $O(n^2)$. We include each substitution job j that can be freely returned with the total cost bounded by $O(n^2 \log n)$. If for none of the pending jobs there could have been formed a feasible set, we verify whether there exists an absorbent job in S^t and carry out the corresponding substitution processing the jobs in the same order as above. The cost of this is $O(n)$. We omit $i(t)$ if none of the above can be done. In addition to the above operations, we recalculate the set of pending gaps of each pending job by generating the corresponding ED-schedule, that takes time $O(n \log n)$ for each job. Hence, the total cost at each scheduling time t is $O(n^2 \log n)$, and since the total number of scheduling times is $O(n)$, the overall cost is $O(n^3 \log n)$. $\square$

Acknowledgements Author is grateful to Christoph Durr for sharing time and discussions.

References

1. P. Baptiste. "An $O(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs". *Operations Research Letters* 24, 175-180 (1999)
2. E.L. Lawler. "A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs". *Annals of Operations Research* 26, 125-133 (1990)

Solving Examination Timetabling Problems using Honey-bee Mating Optimization (ETP-HBMO)

Nasser R. Sabar • Masri Ayob • Graham Kendall

Abstract Examination timetabling problems deal with assigning a set of exams into a limited number of timeslots, whilst satisfying a set of constraints. In this work, we propose ETP-HBMO (Examination Timetabling Problems-Honey-bee Mating Optimization) algorithm for solving examination timetabling problems. The honey-bee mating process is considered as a typical swarm-based approach to optimization, in which the search algorithm is inspired by the process of real honey-bee mating. The mating process (i.e. generating a new solution) of the queen (current best solution) begins when the queen leaves the nest to perform a mating flight during which time the drones (trial solutions) follow the queen and mate with her. In this work, we test ETP-HBMO on Carter's un-capacitated benchmark examination timetable dataset and evaluate the performance using standard proximity costs. Results demonstrate that the performance of the Honey-bee Mating Optimization Algorithm is comparable with the results of other methodologies that have been reported in the literature over recent years. Indeed, ETP-HBMO outperformed other approaches on a few instances. This indicates that ETP-HBMO is effective in solving examination timetabling problems, and worthy of further research.

1 Introduction

Since exam timetabling problems occur frequently throughout an institution's academic calendar, it is a problem that needs to be tackled on a regular basis. Examination timetabling problems can be defined as an allocation of a number of exams to a given number of time periods (see Carter [1]). The allocation is subject to sets of hard and soft constraints. Hard constraints cannot be violated under any circumstances. For example, any exams which involve common students cannot be scheduled at the same time (conflicting exams). A feasible timetable is one in which all examinations have been assigned to feasible timeslots without violating any hard constraints. Soft constraints can be violated if absolutely necessary but every effort is made to satisfy them as far as possible. However, it is usually impossible to satisfy all the soft constraints. For example, we attempt to spread conflicting exams as far apart as possible to allow sufficient time between exams so that students have enough revision time between exams. Unfortunately, it is not always possible to provide as much time as students would ideally want. In order to minimize violations of soft constraints, a penalty cost is attached to each violation as a deterrent measure. The quality of each examination timetable is

Nasser R. Sabar

Jabatan Sains Komputer, Fakulti Teknologi dan Sains Maklumat, Universiti Kebangsaan
Malaysia, 43600 UKM, Bangi Selangor
E-mail: naserdolayme@yahoo.com

Masri Ayob

Jabatan Sains Komputer, Fakulti Teknologi dan Sains Maklumat, Universiti Kebangsaan
Malaysia, 43600 UKM, Bangi Selangor
E-mail: masri@ftsm.ukm.my

Graham Kendall

ASAP Research Group, School of Computer Science, The University of Nottingham,
Nottingham NG8 1BB, UK.
E-mail: gxk@cs.nott.ac.uk

ultimately measured according to the value of penalty costs incurred. The lower the penalty, the better the quality of the exam timetable.

Over the last decade, many university exam timetabling papers have appeared in the literature. Many meta-heuristics approaches have been reported for solving examination timetabling problems. For example: graph based heuristics [2] [3], basic local search [4] [5], hybrid algorithm [6], tabu search [7] [8], simulated annealing [9], genetic algorithms [10], memetic algorithms [11] [12] great deluge algorithms [13], Ant Colony [14], particle swarm optimization [15] and Fuzzy Reasoning [16]. For interested reader, there are also a number of surveys and papers on exam timetabling [17-24].

However, as far as we are aware, there has been no work undertaken to solve examination timetabling problems utilizing a Honey-bee Mating Optimization Algorithm. This work proposes the use of such an algorithm which is tested on the Carter benchmark datasets and the results are compared with other methodologies. We demonstrate how an intelligent technique inspired by nature can be used to give good quality solutions. In order to evaluate the efficiency of the Honey-Bee Mating Optimization Algorithm, we test Honey-Bee Mating Optimization on the un-capacitated Carter's benchmark examination timetable dataset [25] (type I, [18]) and evaluate the quality using standard proximity cost [25].

2 Honey -Bees Mating Optimization Algorithm

The Honey-bee Mating Optimization Algorithm was proposed by Abbass [26] [27] in 2001. Since then it has been used for a number of different applications where it was found to outperform some better known algorithms for different applications such as: Job Shop Scheduling, Integrated Partitioning/Scheduling, Data Mining, 3-sat problem, water resources, management problems, Nonlinear constrained and Unconstrained optimization, Stochastic Dynamic Programming, Continuous Optimization, Data Mining –Clustering, and Genetic Algorithm Improvement (see [28]). However, as far as we aware, it has not been applied to examination timetabling problems. A honey-bee colony consists of the queen(s), drones, worker(s), and broods. The Honey-bee Mating Optimization Algorithm mimics the natural mating behavior of the queen bee when she leaves the hive to mate with drones in the air ([27] [29]). Each time a mating takes place, the genetic pool is enhanced to by adding sperm to the spermatheca. A heuristic (worker) is introduced to the original HBMO to improve the queen's genetic constitution. Before the mating flight begins, the queen is initialized with some energy and only ends her mating flight when the energy level drops below the threshold (which is close to zero) [29]. A queen mates with a drone probabilistically using an annealing function (Eq.1) ([26], [27]):

$$P(Q, D) = e^{\left[\frac{-\Delta(f)}{Speed(t)} \right]} \quad (1)$$

In the above equation, $P(Q, D)$ represents the probability of the drones successful mating. $\Delta(f)$ represents the absolute difference between the drone fitness which constitutes the trial solution and the queen's fitness or best known solution ($\Delta(f) = [F(Q) - F(D)]$). $Speed(t)$ refers to the queen's speed at the time of mating (time t). However, according to the annealing function, the queen's speed is high at the beginning of her flights; therefore the probability of mating is high, as it is when the fitness of the drone is as good as the queen's. As the mating flight continues the queen's speed and energy decays according to equations 1 and 2.

$$\left. \begin{aligned} Speed(t+1) &= \alpha \times Speed(t) \quad (2) \\ energy(t+1) &= \alpha \times energy(t) \quad (3) \end{aligned} \right\} \text{Where } t \in [0, 1, 2 \dots T]$$

α is a factor $\in [0, 1]$, representing the *decay rate* which relates to the rate of energy and speed reduction after each transition in the mating process. Initially, the queen's speed and energy level is randomly generated (speed(0) is the initial speed). Then, a number of mating flights are undertaken. The queen moves between different states (solutions) in the allocated space according to her speed and mates with drones using Eq. 1. Once a drone passes the probabilistic decision rule and has mated successfully with the queen its sperm is added to the queen's spermatheca to generate a list of partial solutions and the queen's speed and energy are reduced according to Eq.2 and Eq.3 respectively. At the end of the mating flight the queen returns to the nest, randomly selects sperms and performs crossover and mutation to produce a brood which is enhanced by a worker (heuristic). The numbers of workers generated for the algorithm represent the number of heuristics encoded in the program. Afterwards, if the fittest brood is found to be better than the queen it replaces her, the remaining broods are destroyed and then another mating flight is initiated. The HBMO algorithm consists of three user-defined parameters namely. 1) Fertilization capacity, which corresponds to the size of the queen's spermatheca, and represents the maximum number of possible matings during a single flight. After each successful mating, the genotype of the drone is added to queen's spermatheca and the queen's spermatheca size is increased by one until the size of spermatheca is reached. 2) The number of queens to be mated. 3) The number of broods produced by the queen as a result of the mating. Figure 1 show the original Honey-bee algorithm (adopted from Abbass [27]):

```

Initialize workers
Randomly generate the queens
Apply local search to get a better queen
For a pre-defined maximum number of mating-flights.
    For each queen in the queen list
        Initialize energy, speed and position
        While the queen's spermatheca is not full and energy >0
            The queen moves between states and probabilistically chooses drones
            If a drone is selected, then
                Add its sperm to the queen's spermatheca
            End if
            Update the queen's internal energy and speed
        End while
    End for each
Generate broods by crossover and mutation
Use workers to improve broods
Update workers' fitness
While the best brood is better than the worst queen
    Replace the least-fittest queen with the best brood
    Remove the best brood from the brood list
End while
Kill all broods

```

Fig. 1 –Original HBMO-[27]

3 ETP-HBMO

In this work, we propose a variation of HBMO that is ETP-HBMO for solving examination timetabling problems. ETP-HBMO combines a number of different procedures. Each one represent to a different phase of the Honey-bee mating process. Firstly, we select the number of Honey-bees that would make up the population of the initial hive. In this work, the initial population was created by utilizing graph coloring heuristics (hybridizations of *Least Saturation Degree*, *Largest Degree first* and *Largest Enrolments First* heuristics) to construct initial solutions (see Ayob et al. [30]). The queen was chosen according to the best solution we obtained from this initial hive. The other solutions generated during this phase became the drones to be used during the mating flight (trial solutions). Secondly, we used the probabilistic rule in Eq.1 to determine which drone the queen mated with. If the mating is successful (i.e., the drone passes the probabilistic decision rule), the drone's sperm is stored in the queen's spermatheca (mating pool). Then, by crossing the drone's genotypes with the queen's, a new brood (trial solution) is formed which can be improved by conducting a local search, carried out by the workers. In this work, TEMCQ (Tabu Exponential Monte-Carlo with Counter Heuristic for Examination Timetabling (see Sabar et al. [31])) algorithm is used as a local search on broods (new trial solutions). In the Honey-bee Mating Optimization Algorithm the workers improve the broods produced from the mating flight of the queen by initiating local search procedures. Each brood chooses one worker to be subject to a local search phase, with the possibility of replacing the queen if the solution is better than the current queen. If a brood fails to replace the queen, it will become one of the drones in the queen's next mating flight. The major difference between our proposed ETP-HBMO algorithm and the original one is that the drones that have been used are then discarded and the new broods are modified to provide fresh drones for the next mating flight. This ensures that no drone's sperm can be used more than once. Table 1 shows a summary of some of the differences between our ETP-HBMO and the original one. The algorithm is shown in Figure 2 and discussed after.

TABLE 1
TABLE 1: DIFFERENCES BETWEEN OUR METHOD AND ORIGINAL ONE

Parameters	Abbass [27] (original)	Our Proposed ETP-HBMO
Drones generations methods	Drones generated randomly	Drones generated by (LS+LD+LE)
No. of queen	1	1
Local search	Greedy SAT (GSAT)	TEMCQ
mutation	Flip-flop	-

Examination Timetabling Problems-Honey-Bees Mating Optimization
(ETP- HBMO)

1-Initialization

- Define the solutions representations
- Define D, M to be population's size (number of Drones), maximum number of mating flights (number of iteration) respectively.
- Generate the initial population of the bees using (LSD+LD+LE) sol_i ($i=1 \dots D$).
- Calculate the fitness value for each Drones $f(sol_i)$ ($i=1 \dots D$).
- Select the best (sol_i) bee and set it as the queen Q (best solution).
- Set Iter=0; W=1 number of workers; queen's spermatheca_size=0; queen_spermatheca_Max=4; population_drones_size=0; population_drones_size_Max; spermatheca_pool=0; Q=1;

2- While Iter $\leq$ M

- initialize energy = rand [0.5, 1], speed= rand [0.5, 1], t=0; $\alpha=0.9$ // decay rate
- while energy > 0 || queen's spermatheca_size < queen_spermatheca_Max
 Select a drone sol_i from the initial populations
 Calculate $\Delta(f) = f(Q) - f(sol_i)$
 Generate R random number= [0, 1];
 If the exp ($\Delta - (f) / \text{Speed}(t)$) > R // the drone passes the probabilistic condition
 Add sperm of the drone sol_i in the spermatheca_pool (mating pool)
 Queen's spermatheca_size = queen's spermatheca_size +1;
 End if
 t=t+1; energy (t) = $\alpha \times \text{energy}(t-1)$; Speed (t) = $\alpha \times \text{Speed}(t-1)$;

End while

3-For j = 1 to queen's spermatheca_size do

- Select a Drone sperm (sol_j) from queen's spermatheca_pool;
- Generate a brood (sol_j^*) by crossing the Queen's (Q) genotype with the selected Drone sperm (sol_j)
- Initiate the worker (TEMCQ). // set local search parameters.
- Use the selected worker (TEMCQ) to reduce the brood's (sol_j^*) fitness.
- If the brood's $f(sol_j^{**})$ fitter than the queen's $f(Q)$ then
 Replace the queen with the brood, $Q \leftarrow sol_j^{**}$, $f(Q) \leftarrow f(sol_j^{**})$.
- Else
 Add sol_j^{**} to the population_broods_size.
- End if

End for j

4- If Iter == Max then Exit; // terminate the programs

Else

- Iter=Iter + 1;
- Generate new drones (sol_i) by modify broods using different neighbour structures (m1, m2, m3).
- Kill all old drones and insert the new modified broods into queen's spermatheca_pool.

End else

End While

Return the Queen (Best Solution Found)

Fig.2 ETP-HBMO pseudo-code

Step 1: Initialization

A chromosome is used to represent a candidate solution sol_i to the problem where each gene in the chromosome represents a timeslot of the candidate solution. Figure 3 provides an example of the chromosome encoding:

T1	T2	T3	T4	T5	T6	T7	T8	T9
e ₂	e ₁₁	e ₈	e ₁₉	e ₇	e ₁₄	e ₁₆	e ₁₀	e ₇
e ₃	e ₆	e ₉	e ₄	e ₁		e ₁₇	e ₁₂	e ₁₈
	e ₇		e ₁₅	e ₁₃			e ₂₀	e ₅
	e ₁₂		e ₁₈				e ₂₁	

Fig 3: Chromosome (complete timetable)

The algorithm starts with three user-defined parameters and one pre-defined parameter: number of workers $W=1$, that is, the number of heuristics encoded in the program. We use one heuristic search TEMCQ. However, the pre-defined parameter may also be used to alter the number of active heuristics if required which means that the user may choose any of them if there is more than one heuristic encoded in the algorithm. The number of queens ($Q=1$), and the queen's spermatheca (mating pool) size=4, which represents the maximum number of broods that can be produced by all queens. Then we define number of mating flights M , number of broods=4 and the number of drones=12. Then, we create a set of initial solutions by employing (LSD+LD+LE), sol_i , ($i=1 \dots m$). After a set of initial solutions has been generated, the initial solutions are ordered according to their fitness value and the best one is set as the queen Q .

Step 2: Mating flight

We used Eq.1 to select the set of drones sperm (trial solutions) from the initial population to make a mating pool for possible information exchange between the best present solution (queen) and the selected sperm (trial solution). If the drones are accepted we add it to spermatheca pool.

Step 3: Breeding process

A new set of broods (new solutions) can be generated by employing pre-defined crossover operators between the queen (best solution) and drone (trial solutions) (e.g. sol_i). In this work we use a simple crossover (haploid crossover) operation as illustrated in the example shown in Figure 4.

T1	T2	T3	T4	T5	T6	T7	T8	T9
e ₂	e ₁₁	e ₈	e ₁₉	e ₇	e ₁₄	e ₁₆	e ₁₀	e ₁₂
e ₃	e ₆	e ₉	e ₄	e ₁		e ₁₇	e ₁₂	e ₁₈
	e ₇	e ₁₅		e ₁₃			e ₂₀	e ₅
							e ₂₁	

A: Drone

T1	T2	T3	T4	T5	T6	T7	T8	T9
e ₁₃	e ₈	e ₅	e ₁₉	e ₇	e ₁₄	e ₁₈	e ₁₀	e ₇
e ₃	e ₆	e ₉	e ₄	e ₁	e ₁₂	e ₂	e ₁₂	
e ₂₁		e ₁₁	e ₁₅	e ₂₀	e ₁₆		e ₁₇	

B: Queen

T1	T2	T3	T4	T5	T6	T7	T8	T9
e ₁₃	e ₈	e ₅	e ₁₉	e ₇	e ₁₄	e ₁₈	e ₁₀	e ₇
e ₃	e ₆	e ₉	e ₄	e ₁	e ₁₂	e ₂	e ₁₂	
e ₂₁		e ₁₁	e ₁₅	e ₂₀	e ₁₆		e ₁₇	
e ₈							e ₁₆	
e ₉							e ₁₇	
e ₁₅								

C: brood

Fig 4: Example of haploid crossover

Two solutions are involved to produce a brood, which are a drone and a queen as shown in Figure 4(a, b). We apply haploid crossover after selecting two random genes (timeslots) from both the drone and the queen. The result of crossover might produce an infeasible solution since some of exams have been repeated. A repair mechanism has to be applied to ensure that repeated exams are removed from the brood as shown in Fig 4(c). The fitness of the trial solutions (broods) might be better than the best solution (queen). Afterwards, a local search is applied to improve the trial solutions (broods). The output from the local search is compared with the best solution. If it is better, the queen is replaced by the brood (new solution). Otherwise we keep the original queen as the best solution. The new brood will be modified using the procedure shown in Figure 5, in order to generate new drones to replace the old drones for the next mating flight.

```

-Define the number_of_broods to be modified

For i=1 to number_of_broods do
    • Randomly move one e1 exam to another timeslots
      with ensuring the feasibility.
    • Swap two exams e1, e2 between different timeslots
      with ensuring the feasibility
    • Permute three exams e1, e2, and e3 in 3- cycles
      with ensuring the feasibility.
End of for
    
```

Fig. 5 Modified procedure pseudo-code

Step 4: Check the termination criteria

The termination criteria for ETP-HBMO algorithm is a number of mating flights.

4 Experiments and Results

In order to evaluate the performance of the ETP-HBMO algorithm, we used Carters uncapacitated exam timetabling benchmark datasets and proximity cost [25] (type I, [18]). We performed our experiments on 12 datasets (10 runs for each dataset, each run taking 2-8 hours based on the size of the dataset). The characteristics of the datasets are shown in Table 2.

The proposed algorithm was implemented in Visual C++ 6.0 on a PC AMD Athlon with a 1.92 GHz processor and 512 RAM running Windows XP 2002. In this work, we have tested a number of different parameter settings and the ones selected are those values that gave the best results with respect to solution quality and the computational time needed to achieve this solution. These parameters will be further tuned in our future work. Table 3 presents the parameter values that we have used in the current implementation

The average and best (best result out of ten runs) values are presented in Table 4. The best results obtained from literature and our works are shown in bold. Result show that ETP-HBMO is able to outperform other approaches that have been reported in the literature six instances (Car-f-92, Kfu-s-93, Sta-f-83, Tre-s-92, Uta-s-92, and Yor-f-83).

In Fig. 6, we show the box-plots of solution quality distributions. In most of cases our approach is able to produce robust solutions.

TABLE 2

UN-CAPACITATED STANDARD CARTER BENCHMARK EXAM TIMETABLING DATASET

Data sets	Number of timeslots	Number of examinations	Number of Students
Car-f-92	32	543	18419
Car-s-91	35	682	16925
Ear-f-83	24	190	1125
Hec-s-92	18	81	2823
Kfu-s-93	20	461	5349
Lse-f-91	18	381	2726
Rye-s-93	23	486	11483
Sta-f-83	13	139	611
Tre-s-92	23	261	4360
Uta-s-92	35	622	21267
Ute-s-92	10	184	2750
Yor-f-83	21	181	941

TABLE 3

ETP-HBMO PARAMETERS

No	Parameters	Value
1-	No. of Queen	1
2-	No. of Drones	12
3-	No. of Mating Flights	1000
4-	Size of Queen Spermatheca	4
5-	No. of Broods	4
6-	No. of Workers	1
7-	No. of selected gens in Crossover	8
8-	α decay rate	0.9
9-	Speed and energy	[0,1]
10-	No. of iteration for TEMCQ	5000
11-	Q TEMCQ	10
12-	Tabu Tuner TEMCQ	4

TABLE 4
RESULTS OBTAINED FROM ETP-HBMO COMPARED TO THE LITERATURE

Data sets	Carter et al. [24]	Di Gaspero and Schaerf [7]	Caramia et al. [5]	Burke and Newall [4]	Merlot et al. [6]	Kendall and Hussin [8]	Asmuni et al. [16]	White et al. [23]	Burke et al. [17]	Burke et al. [2]	Burke et al. [21]	Abdullah et al. [24]	Abdullah et al. [20]	ETP HBMO (Average)	ETP HBMO (Best)
Car-f-92	6.2	5.2	6.0	4.10	4.3	4.67	4.56	4.63	4.4	5.36	4.0	4.4	4.1	4.30	3.90
Car-s-91	7.1	6.2	6.6	4.65	5.1	5.37	5.29	5.73	4.8	4.53	4.6	5.2	4.8	4.86	4.79
Ear-f-83	36.4	45.7	29.3	37.05	35.1	40.18	37.02	45.8	35.4	37.92	32.8	34.9	36.0	36.43	34.69
Hec-s-92	10.8	12.4	9.2	11.54	10.6	11.86	11.78	12.9	10.8	12.25	10.0	10.3	10.8	10.84	10.66
Kfu-s-93	14.0	18.0	13.8	13.90	13.5	15.84	15.81	17.1	13.7	15.2	13.0	13.5	15.2	13.41	13.00
Lse-f-91	10.0	15.5	9.6	10.82	10.5	-	12.09	14.7	10.4	11.33	10.0	10.2	11.9	10.56	10.00
Rye-s-93	7.3	-	6.8	-	8.4	-	10.35	11.6	8.9	-	-	8.7	-	11.90	10.97
Sta-f-83	161.5	160.8	158.2	168.73	157.3	157.38	160.42	158	159.1	158.19	159.9	159.2	159.0	159.67	157.04
Tre-s-92	9.6	10.0	9.4	8.35	8.4	8.39	8.67	8.94	8.3	8.92	7.9	8.4	8.5	8.00	7.87
Uta-s-92	3.5	4.2	3.5	3.20	3.5	-	3.57	4.44	3.4	3.88	3.2	3.6	3.6	3.28	3.10
Ute-s-92	25.8	29.0	24.4	25.83	25.1	27.60	27.78	29.0	25.7	28.01	24.8	26.0	26.0	26.98	25.94
Yor-f-83	41.7	41.0	36.2	37.28	37.4	-	40.66	42.3	36.7	41.37	37.28	36.2	36.2	36.77	36.18

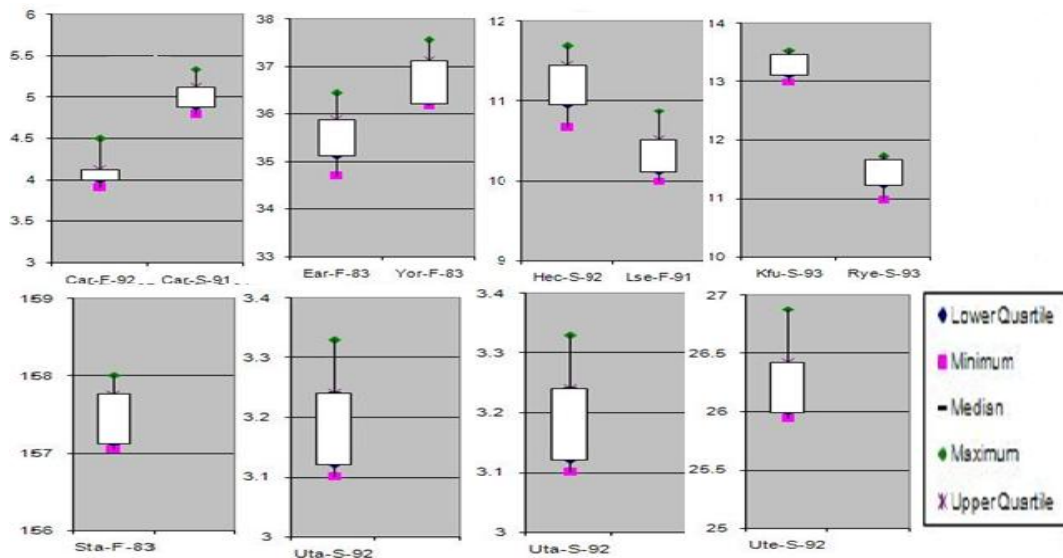


Fig 6: shows show the box-plots of solution quality distributions

4 Conclusion

In this work, a proposed Honey-bee Mating Optimization algorithm (ETP-HBMO) for solving examination timetabling problems has been presented. It has been previously been applied to other NP-hard problems but this is the first time, as far as we are aware, that it has been applied to examination timetabling. The algorithm mimics the mating flight of a queen bee. A mating flight is considered as a set of transitions, in which the queen mates with drones, probabilistically. At the start, the queen's speed is high and therefore the probability of mating is also high, which is also the case when the fitness of the drones is as good as the queen's. The

algorithms start by initializing the population and the best solution is selected as the queen. Next, a number of matings is performed. In each one, the queen's energy and speed are randomly generated. The queen then moves between different solutions in the solution space, according to her speed, and mates with the drones. If a mating is successful, the drone deposits its sperm in her spermatheca, thus giving her many genotype combinations in order to create a new brood. The worker (heuristic) is then used to improve the brood. Now the queen is replaced with the fittest of the brood. The remaining drones are then killed and the new brood is modified to be new drones for the next mating flight.

The results show that the ETP-HBMO can produce good quality solutions for benchmarks examination timetabling problems. In future work, we plan to widen our investigations to the new benchmark datasets that have recently been introduced; as well as other domains such as course timetabling.

REFERENCES

1. Carter, M.W., Laporte, G.: Recent developments in practical examination timetabling. In: Burke, E.K., Ross, P. (eds.) PATAT 1995. LNCS, vol. 1153, pp. 3–21. Springer, Heidelberg (1996).
2. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A Graph-Based Hyper-Heuristic for Educational Timetabling Problems. *European Journal of Operational Research* 176, 177–192 (2007).
3. de Werra, D.: An introduction to timetabling. *European Journal of Operational Research* 19, 151–162 (1985).
4. Burke, E.K., Newall, J.: Enhancing timetable solutions with local search methods. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 195–206. Springer, Heidelberg (2003).
5. Caramia, M., Dell'Olmo, P., Italiano, G. F.: New algorithms for examination timetabling. In: Naher, S., Wagner, D. (eds.) WAE 2000. LNCS, vol. 1982, pp. 230–241. Springer, Heidelberg (2001).
6. Merlot, L.T.G., Borland, N., Hughes, B.D., Stuckey, P.J.: A hybrid algorithm for the examination timetabling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 207–231. Springer, Heidelberg (2003).
7. Di Gaspero, L., Schaerf, A.: Tabu search techniques for examination timetabling. In: Burke, E.K., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 104–117. Springer, Heidelberg (2001).
8. Kendall, G. and Hussin, N.M.: Tabu Search Hyper-Heuristic Approach to the Examination Timetabling Problem at University Technology MARA. In: Burke, E.K., Trick, M. (eds.) PATAT 2004. LNCS, vol. 3616, pp. 199–218. Springer, Heidelberg (2005).
9. Thompson, J., Dowsland, K.: A robust simulated annealing based examination timetabling system. *Computers Operations Research* 25, 637–648 (1998).
10. Burke, E.K., Elliman, D.G., Weare, R.F.: A hybrid genetic algorithm for highly constrained timetabling problems. In: *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA'95)*, pp. 605–610, Morgan Kaufmann, San Francisco, CA, USA (1995).
11. Burke, E.K., Newall, J.P., Weare, R.F.: A memetic algorithm for university exam timetabling. In Burke, E.K., Ross, P. (eds.) PATAT 1996. LNCS, vol. 1153, pp. 241–250. Springer, Heidelberg (1996).
12. Burke, E.K., Landa Silva, J.D.: The design of memetic algorithms for scheduling and timetabling problems. In: Hart, W. E., Krasnogor, N., Smith, J. E. (eds.) *Studies in fuzziness and soft computing: Recent advances in memetic algorithms and related search technologies*, vol. 166, pp. 289–312. Springer, Heidelberg (2004).
13. Burke, E.K., Bykov, Y., Newall, J.P., Petrovic, S.: A time-predefined local search approach to exam timetabling problems. *IIE Transactions on Operations Engineering* 36, 509–528 (2004).

14. Dowsland, K., Thompson, J.: Ant colony optimisation for the examination scheduling problem. *Journal of the Operational Research Society* 56, 426–439 (2005).
15. Chu, S.C., Chen, Y.T., Ho, J.H.: Timetable scheduling using particle swarm optimization. In: *First international conference on innovation computing, Information and control*. vol.3, pp. 324–327. IEEE Computer Society, Washington, DC, USA (2006).
16. Asmuni, H., Burke, E.K., Garibaldi, J.M., McCollum, B.: Fuzzy multiple ordering criteria for examination timetabling. In: Burke, E.K., Trick, and M.A. (eds.) *PATAT 2004*. LNCS, vol. 3616, pp. 334–353. Springer, Heidelberg (2005).
17. Burke, E.K., Kingston, J., de Werra, D.: In: Gross, J., Yellen, J. (eds.) *Applications to timetabling*. *Handbook of Graph Theory*, pp. 445–474. Chapman Hall/CRC Press (2004).
18. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G., Lee, S.Y.: A Survey of Search Methodologies and Automated System Development for Examination Timetabling. *Journal of Scheduling*, 12(1), 55–89 (2009).
19. Burke, E.K., Petrovic, S.: Recent research directions in automated timetabling. *European Journal of Operational Research* 140, 266–280 (2002).
20. Abdullah, S., Burke, E.K.: A Multi-start Large Neighbourhood Search Approach with Local Search Methods for Examination Timetabling. In Long, D., Smith, S.F., Borrajo, D., McCluskey, L. (eds.) *ICAPS 2006*, pp. 334–337, Cumbria, UK (2006).
21. Burke, E.K., Eckersley, A.J., McCollum, B., Petrovic S., Qu, R.: Hybrid variable neighbourhood approaches to university exam timetabling. Accepted by *European Journal of Operational Research*, Technical Report NOTTCS-TR-2006-2, School of Computer Science, University of Nottingham, United Kingdom (2009).
22. Ayob, M., Abdullah, S., Malik, A.M.A.: A Practical Examination Timetabling Problem at the Universiti Kebangsaan Malaysia. *International Journal of Computer Science and Network Security*, 7(9), 198–204 (2007).
23. White, G. M., Xie, B. S.: Examination timetables and tabu search with longer-term memory. In Burke, E.K., Erben, W. (eds.) *PATAT 2000*, LNCS, vol. 2079, pp. 85–103. Springer, Heidelberg (2001).
24. Abdullah, S., Ahmadi, S., Burke, E.K., Dror, M.: Investigating Ahuja–Orlins large neighbourhood search for examination timetabling. *OR Spectrum* 29, 351–372 (2007).
25. Carter, M.W., Laporte, G., Lee, S.Y.: Examination timetabling: Algorithmic strategies and applications. *Journal of Operational Research Society* 47(3), 373–383 (1996).
26. Abbass, H.A.: A monogenous MBO approach to satisfiability In: *Proceeding of the International Conference on Computational Intelligence for Modeling, Control and Automation, CIMCA'2001*. Las Vegas, NV, USA (2001).
27. Abbass, H.A.: Marriage in honey-bee optimization (MBO): A haplometrosis polygynous swarming approach. *CEC2001*. pp. 207–214. Seoul, Korea (2001).
28. Baykasoğlu, Lale Özbakır and Pınar Tapkan, Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem, Source. In: Felix T. S. Chan and Manoj Kumar Tiwari (eds.) *Swarm Intelligence: Focus on Ant and Particle Swarm Optimization*. ISBN 978-3-902613-09-7, pp. 532, December 2007, Itech Education and Publishing, Vienna, Austria (2007).
29. Afshar, A., Haddad, Bozog, O. Marino, M. A., Adams, B. J.: Honey-bee mating optimization (HBMO) algorithm for optimal reservoir operation. *Journal of the Franklin Institute* 344, 452–462 (2007).
30. Ayob, M., Malik, A.M.A., Abdullah, S., Hamdan, A.R., Kendall, G., Qu, R.: Solving a Practical Examination Timetabling Problem: A Case Study. In: Gervasi, O., Gavrilova, M. (Eds.) *ICCSA 2007*, LNCS, vol. 4707, pp. 611–624. Part III. Springer, Heidelberg (2007).
31. Sabar, N. R., Ayob M. and Kendall G.: Tabu Exponential Monte-Carlo with Counter Heuristic for Examination Timetabling. In: *proceedings of 2009 IEEE Symposium on Computational Intelligence in Scheduling (CISched2009)*, pp. 90–94, Nashville, Tennessee, USA (2009).

Adaptive Selection of Heuristics within a GRASP for Exam Timetabling Problems

Edmund K. Burke · Rong Qu · Amr Soghier

Abstract In this paper, we describe the development of a Greedy Random Adaptive Search Procedure (GRASP) where two low-level graph heuristics, Saturation Degree (SD) and Largest Weighted Degree (LWD) are dynamically hybridised in the construction phase to construct solutions for exam timetabling problems. The problem is initially solved using an intelligent adaptive LWD and SD graph hyper-heuristic which constructs the restricted candidate list (RCL) in the first phase of GRASP. It is observed that the size of the RCL used in each iteration affects the quality of the results obtained. In addition, the SD heuristic is essential to construct a RCL which leads to a feasible solution. However, SD does not perform well at the early stages of the construction. Therefore, LWD is used until a certain switching point is reached. The hyper-heuristic adaptively determines the size of the RCL in each iteration and the best switching point after evaluating the quality of the solutions produced. In the improvement phase of GRASP, it is observed that tabu search slightly improves the constructed solutions when compared to steepest descent but it takes a longer time. The approach adapts to all the benchmark problems tested. The comparison of this approach with state-of-the-art approaches indicates that it is a simple yet efficient technique. The results also indicate that the technique could adapt itself to construct good quality solutions for any timetabling problem with similar constraints.

1 Introduction

For the past ten to fifteen years, meta-heuristics have strongly influenced the development of modern search technology at the inter-disciplinary interface of Artificial Intelligence and Operational Research [4]. Meta-heuristics can be applied in a wide variety of different areas

Edmund K. Burke
School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK
E-mail: ekb@cs.nott.ac.uk

Rong Qu
School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK
E-mail: rxq@cs.nott.ac.uk

Amr Soghier
School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK
E-mail: azs@cs.nott.ac.uk

such as scheduling, data mining, cutting and packing, bio-informatics and many others. This facilitated the understanding of different problems in depth and the development of intelligent systems which would automatically solve large complex problems. Many of the meta-heuristic approaches which have been developed are particularly problem-specific. This is appropriate in certain circumstances but there are scenarios where we require a more generic methodology. This motivates the idea of designing automated systems that can operate on a range of related problems rather than on just one particular problem [5].

Hyper-heuristics represent a promising search technology that is concerned with improving the generality of optimisation systems. This area involves the design of intelligent systems that have the ability to decide automatically upon the appropriate methodology to be used to solve a certain problem.

University exam timetabling is an extremely important administrative activity that has attracted many researchers over forty years [22]. Many techniques over the years have been based on graph colouring [3]. This is the process of assigning colours to the vertices in a graph so that no adjacent vertices share the same colour. The objective is to use the smallest number of colours possible. Exam timetabling problems with only hard constraints can be represented as graph colouring models. Vertices in the graph represent exams in the problem, and edges represent the conflicts between exams. Colours correspond to time slots. Several heuristics (e.g. [2]) have been developed to solve graph colouring problems. Graph colouring heuristics are still widely used to develop new methods to solve timetabling problems [3].

Greedy Random Adaptive Search Procedure (GRASP) is a meta-heuristic which has attracted significant recent attention [15]. It is a two phase procedure which starts by adaptively creating a randomised greedy solution followed by an improvement phase.

In this paper, we design an intelligent adaptive hyper-heuristic within a GRASP for exam-timetabling problems. Section 2 defines some important terminology and presents background on exam timetabling, hyper-heuristics, graph colouring and GRASP. Section 3 discusses our intelligent adaptive hyper-heuristic using GRASP. The experimental settings and results are presented in section 4. Finally, conclusions are discussed and some future work is proposed in section 5.

2 Background

2.1 The Toronto Benchmark Exam Timetabling Problem

Exam timetabling [22] is the process of assigning a number of exams into a limited number of time slots taking into account a number of constraints. Constraints are divided into two types:

- *Hard Constraints*: These are constraints that cannot be violated under any circumstances.
- *Soft Constraints*: These are constraints that must be satisfied but could be violated. A penalty is added to the evaluation of a solution when these constraints are broken.

We investigate our approach by applying it to the Toronto benchmark exam timetabling problems [10]. This dataset is widely used in exam timetabling research by different state-of-the-art approaches and can therefore be considered as a benchmark [22]. The problem has one hard constraint where conflicting exams cannot be assigned to the same time slot. In addition, a soft constraint is present where conflicting exams should be spread throughout the timetable as far as possible from each other. An objective function is used to evaluate the

quality of the solutions constructed. The goal here is to minimise the cost in this objective function. The quality of a solution is based on the sum of proximity costs given as:

$$\sum_{i=1}^4 \frac{w_i N}{S} \quad (1)$$

where

- $w_i = 2^{5-i}$ is the cost of assigning two exams with i time slots apart. Only exams with common students and are five or less time slots apart are considered as violations
- N is the number of students involved in the conflict
- S is the total number of students in the problem

There has, however, been a problem with these benchmarks. Over the years two slightly different versions of some of the sets have been made available. This has caused some confusion when comparing different approaches in the literature. They were renamed version I and version II in [22] and the issues were thoroughly discussed. In the version II problem instances, 3 instances (sta83 II, yor83 II and ear83 II) contain repetitions in the data files. Therefore, these repetitions were removed [22] and named as version IIc (the “c” stands for “corrected”). A modification to the number of time slots had to be made for sta83 II from 13 to 35 time slots to obtain a feasible solution. The characteristics of the different versions are presented in table 1. The original version I, version II and the version IIc data files are all available at <http://www.asap.cs.nott.ac.uk/resources/data.shtml>.

Problem	Exams I/II/IIc	Students I/II/IIc	Enrolments I/II/IIc	Density	Time Slots
car91 I	682	16925	56877	0.13	35
car92 I	543	18419	55522	0.14	32
ear83 I	190	1125	8109	0.27	24
ear83 IIc	189	1108	8057	0.27	24
hec92 I	81	2823	10632	0.42	18
hec92 II	80	2823	10625	0.42	18
kfu93 I	461	5349	25113	0.06	20
lse91 I	381	2726	10918	0.06	18
sta83 I	139	611	5751	0.14	13
sta83 IIc	138	549	5417	0.19	35
tre92 I	261	4360	14901	0.18	23
uta92 I	622	21266	58979	0.13	35
uta92 II	638	21329	59144	0.12	35
ute92 I	184	2750	11793	0.08	10
yor83 I	181	941	6034	0.29	21
yor83 IIc	180	919	6002	0.3	21

Table 1 Characteristics of the Toronto Benchmark dataset

2.2 Greedy Random Adaptive Search Procedure (GRASP)

Randomly generated solutions are not expected to be of high quality. A greedy function, which does not contain ties or contains deterministic rules to break ties, will always lead to the same solution. To overcome such issues a semi-greedy heuristic such as GRASP can be used [14, 17]. After the candidate list is produced randomly, the best ranked elements are

placed in a restricted candidate list. The element to add to the solution is then randomly chosen. Two different methods were proposed by [14] and [17] to choose the elements to place in the restricted candidate list (RCL). The cardinality-based method chooses a number of elements from the top of the list. The size of the RCL is set through a parameter. The value-based method chooses the elements that fall within a certain threshold (0-100%) from the greedy value. More methods were also proposed in [17].

GRASP can be easily integrated in a hybrid meta-heuristic, as any local search algorithm could be used in the improvement phase. Simulated Annealing and Tabu Search have been used in [11, 12, 19] as local search procedures.

2.3 Hyper-Heuristics

The idea of Hyper-heuristics is to decide the appropriate search method to be used to solve a certain problem. A hyper-heuristic explores the heuristic space instead of exploring the solution space. After choosing certain heuristics to be used, this heuristic combination can be run on the problem and the solution can be evaluated using an objective function. The hyper-heuristic can then use this evaluation to decide whether to accept the current technique or switch to a different one.

Several hyper-heuristics have been recently developed to solve many combinatorial optimisation problems, which include timetabling [6, 7]. Low-level heuristics as employed within hyper-heuristic methods in the literature can be divided into two main categories: Constructive and Improvement.

2.3.1 Hyper-heuristics with Constructive Low-level Heuristics

A hyper-heuristic with low-level constructive heuristics starts with no solution and builds it step by step from the beginning. For example, a hyper-heuristic to solve an exam timetabling problem could initially start with an empty timetable. It then starts creating a full timetable by choosing a certain exam and inserting it in the timetable using a certain heuristic. The task of the hyper-heuristic might be to create a heuristic list which is the combination of heuristics that would allocate all the exams in appropriate time slots. Different heuristic sequences could be created and applied to the problem at hand until all the exams are scheduled and the solution is accepted. During the timetable construction, the hyper-heuristic evaluates the moves and chooses the next heuristic to be applied. In certain cases, the hyper-heuristic could decide to remove exams already scheduled or back track according to the evaluation carried out in each step.

In timetabling, only a few constructive techniques have been investigated (see [7]). A technique employing fuzzy logic to combine graph heuristics to solve exam-timetabling problems was discussed in [1]. In [24], an investigation is made to employ Genetic Algorithms to evolve the configurations of constraint satisfaction methods on constructing problem solutions.

Another approach was developed in [9] where a Case-Based Reasoning system was implemented. The system is used as a heuristic selector for solving exam timetabling problems. Different problem solving situations and their corresponding constructive heuristics are stored in the system. The system then compares the problem situation at hand with the cases that are already stored. The heuristics previously used are retrieved to solve the problem.

In [23], a genetic-based hyper-heuristic is employed using four basic constructive low-level heuristics to solve one-dimensional bin packing problems. Very good results have been found and some potential issues were discussed.

2.3.2 Hyper-heuristics with Low-level Improvement Heuristics

This Hyper-heuristic starts with a complete timetable, which it will then iteratively improve. Several heuristics could then be applied such as moving an exam from one slot to another or swapping two exams. The effect of applying a certain heuristic is then examined and the hyper-heuristic decides whether to approve the move or to perform a different one.

In [18], a tabu list was used to avoid the repetition of bad moves for a certain number of iterations. In [6], a similar approach was applied to several instances of a nurse rostering and a course-timetable problem. It was shown to be competitive with algorithms specifically designed for these problems.

The tabu search hyper-heuristic investigated in [6] inspired the development of a Simulated Annealing method to search a set of low-level heuristics to determine the shipper sizes in transportation problems in [13]. The low-level heuristics here are both neighbourhood structures and sampling policies within the solution space.

In [16], a Genetic Algorithm is used to choose the low-level heuristics to be applied to simulated course scheduling and student project presentation problems. The low-level heuristics here are some move strategies that are applied to the problems.

2.4 The Graph based Hyper-heuristic (GHH)

Several heuristics have been developed to solve graph colouring problems and exam timetabling problems (e.g. [10]). The idea of these heuristics is based on estimating difficulty and colouring the most difficult vertices first in a simple constructive technique. Some graph heuristics in timetabling are presented as follows (These descriptions are taken from [10]):

- *Largest Degree First (LD)*: Exams that have a larger number of clashes (common students) with other exams should be scheduled first.
- *Largest Colour Degree First (CD)*: Exams that have a larger number of clashes with the exams already scheduled should be scheduled first.
- *Saturation Degree (SD)*: This heuristic will order the exams according to the number of time slots available for this exam to be scheduled. The exams with the lowest number of available time slots will be given higher priority.
- *Largest Weighted Degree (LWD)*: This heuristic is similar to LD but uses more information when applied to exam-timetabling problems. It orders the exams that have the largest number of students, involved in clashes with other exams first.
- *Largest Enrolment (LE)*: Largest enrolment orders the exams according to the number of students in each exam.

In a graph based hyper-heuristic for timetabling problems [20], a sequence of graph heuristics are used to order the events (courses or exams) one by one to construct a full timetable. A number of high-level heuristics were used to compare their performance to find the best heuristic sequences to solve exam and course timetabling problems. In each iteration, a heuristic sequence is chosen and used to construct a solution. After a sequence is applied and a solution is generated, this solution is evaluated using the objective function.

A timetable is constructed by adding the events one by one in the available time slots. According to the heuristic sequence being followed, the first event in the ordered list is scheduled to the first time slot that leads to the least cost. The cost in this situation is calculated by the soft constraint violations incurred. After an event is scheduled, the remaining events are re-ordered by using the next heuristic in the heuristic sequence. The process continues until all the events are scheduled and a full timetable is constructed.

An illustrative example of ordering five exams (e1-e5) using two graph heuristics is shown in figure 1. The example assumes that the events are ordered according to their SD and LWD as shown. In the 1st and 2nd iterations, LWD is used (as shown in the heuristic sequence) and therefore e1 and e2 are scheduled. Assuming that the order of the remaining elements does not change after the SD list is updated in the 3rd iteration, e5 is then scheduled. Finally, e4 followed by e3 are scheduled using the LWD ordering.

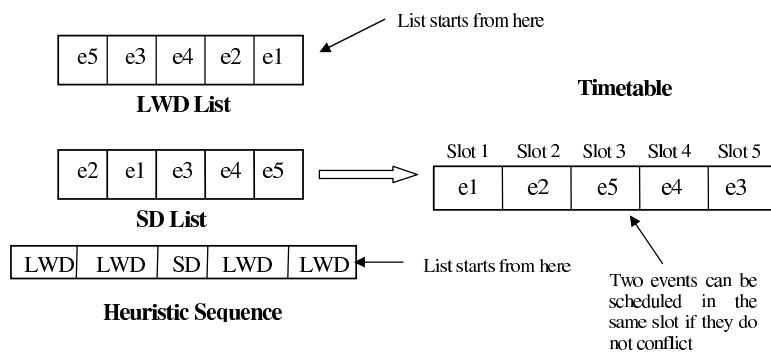


Fig. 1 An illustrative example of using a heuristic sequence to construct a timetable

The quality of the timetable constructed is then evaluated and the cost is compared with any previous timetables produced using other sequences. Certain sequences during constructing the timetable cannot schedule a certain event as it causes conflicts. Hence, the sequence is discarded and the high-level heuristic uses another sequence to construct another solution. In this approach, the high-level heuristic acts on the heuristic sequences and does not act on the problem domain itself. The search is performed on the heuristic space and the high-level heuristic is controlled by the performance of running heuristic sequences to construct timetables. The GHH approach is used to construct a heuristic sequence which will then construct a solution. This technique applies heuristics adaptively to construct solutions and could therefore be generalised to be applied to a set of similar problems (i.e. timetabling problems in this case). In this paper, we present an adaptive approach where two low-level graph heuristics (SD and LWD) are dynamically hybridised in the construction phase of a GRASP. The hyper-heuristic uses SD & LWD to construct the RCL in the GRASP. The exam to be scheduled next is randomly chosen from the RCL. The process continues until all the exams are scheduled or a certain exam causes a conflict. Furthermore, an improvement is made to the constructed timetables in the second phase of GRASP.

3 An Intelligent Adaptive GHH within a GRASP

3.1 An Intelligent Adaptive Hybridisation of SD and LWD within a GRASP

As previously stated, a random ordering method was used in an iterative approach to generate heuristic sequences to be applied on exam and course timetabling problems [20]. Applying random heuristic sequences to a problem is usually a time consuming process. Therefore, an intelligent approach that would always lead to a feasible solution would be preferred. Figure 2 presents the pseudo-code of an intelligent technique to hybridise the SD and LWD graph heuristics using a GRASP.

```

for n = 0 to n = 50          // n : number of iterations
{
    while(number of exams scheduled < total number of exams)
    {
        RclSize = 0.03 x u      // u : number of unscheduled exams
        Create Heuristic Sequence & restricted candidate list of RclSize
        Fill two thirds of the heuristic sequence with the SD heuristic
        Starting from the switching point, fill the rest of the heuristic sequence with LWD
        Run the heuristic sequence to the problem and fill the RCL
        Schedule any exam randomly from the RCL
    }

    if(solution is feasible)
        Evaluate the solution and apply a steepest descent improvement
    else
    {
        Discard the solution & continue without applying an improvement
    }
    //end if

    if(current solution is better than the best solution)
        best solution = current solution
    //end if
}

```

Fig. 2 The pseudo-code of the intelligent hybridised LWD and SD graph based hyper-heuristic within GRASP

It was observed in our previous work [21] that heuristic sequences that combine SD with LWD perform the best for almost all the exam timetabling problems. The motivation of intelligently hybridising LWD and SD comes from the fact that LWD identifies the difficulty of scheduling exams in relation to the rest and SD orders them according to the current state of the available time slots. Therefore, we developed a hyper-heuristic that combines SD and LWD within the construction phase of a GRASP. The GHH is applied to the problem at hand to create a restricted candidate list (RCL) which is used to randomly choose the next exam to be scheduled.

In each iteration, the GHH starts by updating the SD and LWD orderings. The GHH then determines the size of the heuristic sequence required to construct the RCL. The size was initially set to 3% of the number of unscheduled exams. Different RCL sizes were tested

and it was found that using a RCL which has a size of 1%-3% of the unscheduled exams obtains the best solutions according to the size of the problem. Using a RCL of a large size increases the randomness in the solution construction and therefore the choice of the exam to be scheduled next becomes more difficult. In contrast, using a very small RCL would limit the choice of exams and would not lead to a feasible solution. After determining the size of the heuristic sequence, the heuristics to be used are defined. The sequence is run on the problem at hand and the outcome exams are placed in the candidate list.

According to the calculated size of the RCL, the GHH creates the heuristic sequence to construct the RCL. In previous work undertaken in [7,8,10], it was proved that using SD on its own outperforms other graph heuristics and results in a better outcome. This is due to the ability of SD to dynamically adapt itself to the current state of the solution construction as it orders events according to the available valid time slots. Therefore, we decided to let the GHH to construct a heuristic sequence which contains twice as many SD iterations than LWD. Hence, this technique gives the GRASP a higher chance to schedule the exams using SD. The size of the RCL will decrease adaptively as the exams are scheduled. The approach will also automatically schedule the last 20 exams using SD only. In addition, an exam that is on top of the SD and LWD orderings will be placed in the RCL twice and will have a higher chance of being randomly chosen and scheduled.

An illustrative example is given in figure 3 to use a GHH to construct a RCL. The GHH uses SD and LWD as low-level heuristics. The number of unscheduled exams are equal to 100. Hence, a heuristic sequence and RCL of size 3 are created. The sequence generated will contain two SD iterations and one LWD as the number of SD elements have to be twice as many as the LWD elements in the RCL. The heuristic sequence is run and e56 & e6 will be taken from the SD list and placed in the RCL. In addition, e9 will be taken from the LWD list and placed in the RCL. Therefore, the RCL will contain a ratio of 2:1 SD against LWD elements.

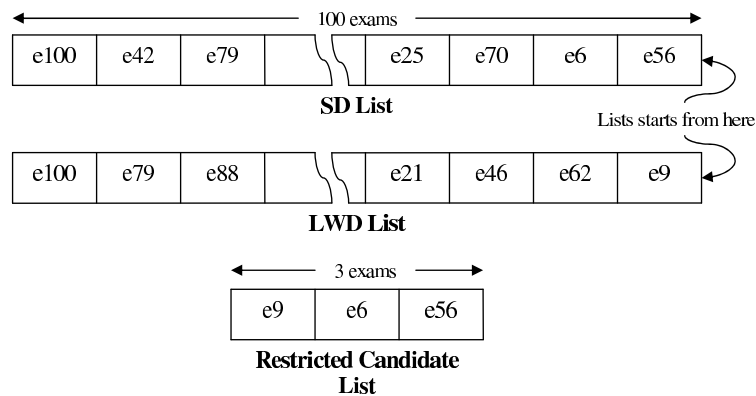


Fig. 3 An illustrative example of building a RCL from SD and LWD

3.2 GRASP & GHH

GRASP is introduced here to find a quick solution that would lead to feasible solutions in the heuristic space. This is done through two steps. The adaptively hybridised LWD & SD

GHH discussed in section 3.1 is used to build the restricted candidate list in the construction phase of GRASP. Subsequently, a random exam is chosen from the RCL and scheduled. This process continues until all the exams are scheduled. Finally, the constructed solution is improved through a local search in the solution space.

3.2.1 GRASP construction phase

In this construction phase, the GHH starts by applying LWD to schedule the first exams in the first iterations. This process is done adaptively to find the switching point which leads to the best solution. The GHH then switches to combine LWD and SD to build the GRASP RCL for the rest of the iterations. A random exam is then chosen from the RCL and scheduled. This process will either schedule all the exams and construct a feasible solution or it will stop at a certain point where the exam cannot be scheduled in any time slot due to its conflict with those already scheduled. If a feasible solution is obtained, the solution is evaluated using the objective function. An improvement is then applied to the constructed solution in the second phase of the GRASP. Otherwise, the process restarts if a feasible solution cannot be obtained.

Since SD orders exams according to the number of time slots available, starting to solve a problem using SD is inefficient as all the time slots would be available [7, 8, 10]. This would result in many ties at the beginning and different approaches could be used to overcome this difficulty.

In view of the fact that LWD orders the exams that have the largest number of students involved in clashes with other exams first, LWD gives priority to the exams that will be difficult to schedule later on in the process. Therefore, we used LWD to order exams in the first iterations before introducing SD.

The hybridised GHH starts by employing LWD. At a certain point, the hyper-heuristic has to switch from LWD to using SD and LWD as described in section 3.1.

We applied the intelligent GHH to the Toronto benchmark exam timetabling problems [10] to adaptively select the best switching point from using LWD to combining both LWD & SD in a GRASP. The following formula was used to choose an initial switching point which is then adaptively tuned:

$$\text{Switching Point} = \frac{\text{Total Number of Exams}}{\text{Total Number of Time slots}} \quad (2)$$

The starting switching point is the number of exams required so that the exams are spread evenly in the time slots available. The hyper-heuristic uses this as the initial switching point and applies the approach described in the previous section to obtain a solution. The switching point is then incremented and decremented by one and the new switching points are used to obtain another set of solutions. The process stops when a feasible solution is not obtained for two consecutive iterations.

Using this adaptive approach, the best switching point is automatically found for different problems. We observed that using LWD on its own will not construct a feasible solution. Furthermore, SD has to be used early in the process to be able to identify exams that will be difficult to schedule later on. In addition, it is observed that using different switching points leads to different solutions. This is due to the fact that SD dynamically orders the exams and therefore the amount of LWD iterations employed at the beginning will affect the decisions made by SD.

3.2.2 GRASP improvement phase

The GRASP improvement phase starts with a feasible solution from the construction phase and a steepest descent hill climbing improvement is employed. The hill climbing is run for $(e \times 10)$ iterations where e is the number of exams. In each iteration a random exam is chosen and the improvement of scheduling this exam in all the time slots of the timetable is observed. The best improvement is stored at the end of the process and the result is obtained.

4 Experimental Results

The adaptive intelligent GHH using GRASP is implemented and tested on a PC with an Intel Pentium 3.4 GHz processor, 1GB RAM and Windows XP. The program is coded using C++. The intelligent hyper-heuristic, the low-level graph heuristics and the GRASP are implemented as objects with a common interface to work together. The hyper-heuristic could be easily applied to different exam timetabling problems in the dataset. The objectives of the experiments are:

- To demonstrate the role of adding a GRASP to a GHH to improve the quality of the solutions obtained (shown in table 4).
- To raise the generality of exam timetabling problem solving approaches through producing a dynamic hyper-heuristic that would adapt to a given exam timetabling problem and generate results which are comparable to the results published in the literature (shown in table 4).
- To compare the quality of this approach with other known published methods (shown in table 4).

4.1 Analysis on the Intelligent Adaptive Hybridisation of the LWD and SD GHH to construct a RCL in a GRASP

In the first set of experiments, the GHH using GRASP is applied to version I and version II of the Toronto benchmark exam timetabling problems to see its ability to construct a feasible solution as well as to evaluate the quality of the solutions produced. The effect of intelligently hybridising different low-level heuristics in a GRASP is investigated. It is observed that SD and LWD performed the best when hybridised together in the RCL rather than using them separately. The GHH chooses the exams which have a high priority in both the SD and LWD orderings to construct the RCL (i.e. the exams that have the largest number of students, involved in clashes with other exams and the exams that have the lowest number of slots available at that time to schedule in).

In the second set of experiments, using different percentages of SD and LWD in the RCL was investigated. It was observed that using the SD heuristic more than the LWD in the RCL produces better results. Therefore, we used a ratio of 2:1 SD to LWD heuristics to construct the RCL. This shows the importance of SD being used as it updates the ordering in each iteration and adapts to the current situation.

Five runs with 100 iterations each using distinct seeds are carried out for all the datasets except for (car92 I), (uta92 I), (uta92 II) and (car91 I) where only three runs were performed. The average costs of the solutions and the average percentage of feasible solutions constructed in the runs are presented in table 2. A comparison with using the same approach with SD and LWD separately is also shown.

Problems	GRASP Construction using SD & LWD			GRASP Construction using only SD			GRASP Construction using only LWD		
	Average Cost	Best Cost	Percentage of feasible solutions	Average Cost	Best Cost	Percentage of feasible solutions	Average Cost	Best Cost	Percentage of feasible solutions
hec92 I	13.00	12.41	35%	13.82	13.02	21%	-	-	0%
hec92 II	13.72	12.37	20%	-	-	0%	-	-	0%
sta83 I	167.31	163.63	19%	174.61	169.10	15%	172.52	171.49	38%
sta83 IIc	40.42	36.82	69%	40.53	38.03	63%	36.03	34.79	69%
yor83 I	43.38	42.54	5%	45.62	44.21	4%	-	-	0%
yor83 IIc	52.57	52.24	3%	-	-	0%	-	-	0%
ute92 I	31.90	30.01	13%	32.78	30.74	10%	-	-	0%
ear83 I	39.06	37.71	32%	45.74	43.94	29%	-	-	0%
ear83 IIc	42.54	40.49	6%	48.27	48.21	3%	-	-	0%
tre92 I	9.38	9.00	39%	10.16	9.30	22%	-	-	0%
lse91 I	12.89	12.29	3%	13.48	12.51	2%	-	-	0%
kfu93 I	17.24	16.84	6%	18.28	18.24	2%	-	-	0%
car92 I	4.84	4.74	3%	5.07	4.85	0%	-	-	0%
uta92 I	3.65	3.62	100%	3.97	3.70	32%	-	-	0%
uta92 II	3.72	3.60	20%	-	-	0%	-	-	0%
car91 I	5.59	5.48	94%	5.78	5.49	57%	-	-	0%

Table 2 Average results, best results and percentage of feasible solutions produced from the GRASP construction phase using SD & LWD in comparison to using them separately. A (-) indicates that no feasible solution could be obtained

Furthermore, another set of experiments is run to observe the effect of changing the size of the restricted candidate list used. Different RCL sizes were tested on problems with different size. The experiments showed that using a RCL with 3% the size of the unscheduled exams produces the best results for small problems (i.e problems with less than 200 exams). For medium sized problems, a RCL with 2% the size of unscheduled exams produced the best results (i.e problems with exams between 200 - 400). For large problems, a 1% produced the best results (i.e. problems with more than 400 exams). A large RCL introduces more randomness in large problems while a small RCL restricts the choice in small problems. In all the cases, the process starts with high randomness which adaptively decreases and becomes more deterministic as exams are scheduled.

Finally, another important factor for this method to construct a feasible solution is the switching point from using LWD to combining both SD and LWD in the GRASP. It is observed that for each problem, more than one switching point between the 1st and the 25th iteration led to a feasible solution. The choice of the switching point that would lead to the best solution is difficult especially when the method is applied to different problems. Using the formula and approach described in section 3.2.1, the switching point problem was adaptively solved. The average percentage of feasible solutions obtained in the runs for each problem is shown in table 2.

4.2 Analysis on GRASP improvement using steepest descent

In the second set of experiments, steepest descent is applied to the constructed solutions to perform a local search on the solution space. The improvement is applied to all the instances where a feasible solution is constructed. It is observed that the solutions with the least cost

from the construction phase do not always lead to the best improvements. This is because the solutions constructed by different heuristic sequences in the heuristic space lead to different neighbourhoods in the solution space being searched to find the global optimum. We also investigated using tabu search and found that the tabu search improvement performs slightly better than steepest descent in some problems. However, the computational time for performing a tabu search improvement is much higher. Therefore, we used steepest descent as it produces competitive results in comparison with tabu search in a shorter time. Table 3 shows the average costs after the improvement and the total average time including construction and improvement.

Problems	GRASP construction Average Cost	GRASP Improvement using Steepest Descent		Total Average Time (sec)
		Average Cost	Best Cost	
hec92 I	13.00	12.14	11.78	83.13
hec92 II	13.72	13.51	12.04	267.50
sta83 I	167.31	166.39	158.94	314.60
sta83 IIc	40.42	38.26	35.19	3376.87
yor83 I	43.38	42.85	42.19	484.58
yor83 IIc	52.24	51.53	50.40	410.77
ute92 I	31.90	31.14	26.62	402.50
ear83 I	39.06	38.71	36.52	5289.05
ear83 IIc	42.54	42.10	39.93	737.30
tre92 I	9.38	9.31	8.99	10817.01
lse91 I	12.89	12.72	12.12	768.33
kfu93 I	17.24	16.64	15.45	748.32
car92 I	4.84	4.67	4.45	8710.55
uta92 I	3.65	3.62	3.50	99053.51
uta92 II	3.72	3.69	3.49	95687.27
car91 I	5.59	5.48	5.37	99874.59

Table 3 Average costs before and after improvement, best costs after improvement and total average time using Steepest Descent in GRASP

4.3 Comparison with state-of-the-art approaches

The best results obtained, the best reported in the literature using different constructive methods [22], the graph based hyper-heuristic [7], the tabu search hyper-heuristic [18] and other methods are presented in table 4. In addition, the standard deviation from the best reported results obtained is shown (σ in table 4).

The results obtained by this approach when compared to the state-of-the-art approaches, indicate the efficiency and generality of the intelligent adaptive hybridisation of LWD and SD GHH. In comparison with the pure GHH approach in [7], the GRASP-GHH performs better in 9 out of the 11 version I cases. Furthermore, it performs better in 7 out of 8 cases reported in [18]. The results of (sta83 II), (yor83 II) and (ear83 II) were not compared to other results in the literature since the methods of modification to deal with the corruption (i.e. repetitions) in the datasets were not stated.

Although the best results reported in the literature were obtained by using different approaches that worked well only on specific instances, the results produced here using the adaptive approach are still in the range of the best-reported results as shown. In addition, the aim of this paper is to illustrate the adaptiveness and the effect of hybridising low-level

heuristics in a hyper-heuristic using GRASP that could be applied to any timetabling problem with similar constraints. The objective here is not to outperform the other approaches that worked only on specific instances. However, we can demonstrate that our approach is competitive to the other approaches in the literature. It was not possible to compare the computational time as the time for most of the approaches was not reported in the literature.

Problems	Steepest Decent in GRASP		GHH Best [7]	Tabu search HH [18]	Constructive Best [22]	Best Reported [22]
	Best Cost	σ				
hec92 I	11.78	1.82	12.72	11.86	10.8	9.2
hec92 II	12.04	0.88	-	-	-	10.8
sta83 I	158.94	1.16	158.19	157.38	158.19	157.3
sta83 IIc	35.19	-	-	-	-	-
yor83 I	42.19	4.24	40.13	-	39.80	36.20
yor83 IIc	50.40	-	-	-	-	-
ute92 I	26.62	0.54	31.65	27.60	25.8	24.4
ear83 I	36.52	5.11	38.19	40.18	36.16	29.3
ear83 IIc	39.93	-	-	-	-	-
tre92 I	8.99	0.78	8.85	8.39	8.38	7.9
lse91 I	12.12	1.78	13.15	-	10.50	9.6
kfu93 I	15.45	1.73	15.76	15.84	14.00	13.0
car92 I	4.45	0.37	4.84	4.67	4.32	3.93
uta92 I	3.50	0.25	3.88	-	3.36	3.14
uta92 II	3.49	0.13	-	-	-	3.40
car91 I	5.37	0.62	5.41	5.37	4.97	4.5

Table 4 Best results obtained by the intelligent adaptive GHH using GRASP compared to the state-of-the-art approaches

5 Conclusions and Future Work

In this paper we present an adaptive approach where two low-level graph heuristics (saturation degree and largest weighted degree) are dynamically hybridised in the construction phase of a greedy random adaptive search procedure (GRASP) for exam timetabling problems. The problem is initially solved using an intelligent adaptive LWD & SD graph hyper-heuristic which constructs the restricted candidate list (RCL) in the construction phase of GRASP. It is observed that the SD heuristic is essential to construct a feasible solution. However, SD does not perform well at the early stages of the construction as the timetable is empty and it returns the same difficulty of scheduling exams at the first few iterations. Therefore, LWD is used until a certain switching point. The switching point is adaptively tuned by the hyper-heuristic after evaluating the quality of the solutions generated.

An improvement is made to the constructed solutions and the best is chosen. Steepest descent is used for the improvement and proved to produce competitive results to a tabu search improvement in a shorter time. It is observed from experiments that the approach adapts to all the problems it is applied to and generates different quality of solutions according to the switching point chosen. Therefore, we created an adaptive adjustment to the switching point according to the quality of the solutions produced.

The comparison of this approach with state-of-the-art approaches indicates that it is a simple yet efficient technique. It is also a more general technique than many in the literature

which can be adapted to construct good quality solutions for any exam timetabling problem with similar constraints.

Future work will include intelligently hybridising more different low-level graph colouring heuristics. In addition, some new adaptive heuristics similar to SD could be designed. Considering the cost of assigning an exam in a certain time slot instead of the first available slot could also improve the quality of the constructed solutions. Tie breakers for the ordered exams could also be considered as well as different switching point selectors.

Finally, one of our primary future research directions is applying the technique developed here to random exam timetabling datasets and evaluating its ability to adapt to the problems being solved. Furthermore, the technique developed could be applied to graph colouring and course-timetabling problems.

References

1. H. Asmuni, E.K. Burke, J. Garibaldi, and B. McCollum. Fuzzy multiple ordering criteria for examination timetabling. In E.K. Burke and M. Trick, editors, *Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling*, volume 3616 of *Lecture Notes in Computer Science*, pages 334–353. Springer, 2004.
2. D. Brelaz. New methods to colour the vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.
3. E.K. Burke, D. de Werra, and J. Kingston. Applications to timetabling. In J. Gross and J. Yellen, editors, *Handbook of Graph Theory*, chapter 5.6, pages 445–474. Chapman Hall/CRC Press, 2004.
4. E.K. Burke and G. Kendall, editors. *Search Methodologies: Introductory Tutorials in Optimisation and Decision Support Techniques*. Springer, 2005.
5. E.K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Meta-Heuristics*, pages 457–474. Kluwer, 2003.
6. E.K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, Dec 2003.
7. E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176:177–192, 2007.
8. E.K. Burke and J. Newall. Solving examination timetabling problems through adaptation of heuristic orderings. *Annals of operations Research*, 129:107–134, 2004.
9. E.K. Burke, S. Petrovic, and R. Qu. Case based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132, 2006.
10. M.W. Carter, G. Laporte, and S.Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of Operational Research Society*, 74:373–383, 1996.
11. R. Colome and D. Serra. Consumer choice in competitive location models: Formulations and heuristics. Technical report, Department of Economics and Management, Universitat Pompeu Fabra, Barcelona, Spain, 1998.
12. H. Delmaire, J.A. Diaz, E. Fernandez, and Ortega M. Reactive grasp and tabu search based heuristics for the single source capacitated plant location problem. *INFOR*, 37:94–225, 1999.
13. K. Dowsland, E. Soubeiga, and E.K. Burke. A simulated annealing hyperheuristic for determining shipper sizes. *European Journal of Operational Research*, Vol 179, issue 3:pp 759–774, 2007.
14. T. Feo and M. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.
15. T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
16. L. Han and G. Kendall. Guided operators for a hyper-heuristic genetic algorithm. In T. D. Gedeon and L. C. C. Fung, editors, *Proceedings of AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference on Artificial Intelligence (AI 03)*, pages 807–820, 2003.
17. J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.
18. G. Kendall and N. Mohd Hussin. An investigation of a tabu search based hyper-heuristic for examination timetabling. In G. Kendall, E. Burke, S. Petrovic, and M. Gendreau, editors, *Selected Papers from MISTA 2003*, pages 309–328. Springer, 2005.

19. M. Laguna and Gonzalez-Velarde. A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent manufacturing*, 2:253–260, 1991.
20. R. Qu and E.K. Burke. Hybridisations within a graph-based hyper-heuristic framework for university timetabling problems. *to appear at Journal of Operational Research Society*, 2009.
21. R. Qu, E.K. Burke, and B. McCollum. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research*, 198(2):392–404, 2009.
22. R. Qu, E.K. Burke, B. McCollum, L.T.G. Merlot, and S.Y. Lee. A survey of search methodologies and automated approaches for examination timetabling. *Journal of Scheduling*, 12(1):55–89, 2009.
23. P. Ross, J.G. Marin-Blazquez, S. Schulenburg, and E. Hart. Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'03)*, pages 1295–1306, 2003.
24. H. Terashima-Marin, P. Ross, and M. Valenzuela-Rendon. Evolution of constraint satisfaction strategies in examination timetabling. In *Proceedings of the Genetic and Evolutionary Conference, Orlando, Florida*, pages 635–642, 1999.

An Extended Great Deluge Approach to the Examination Timetabling Problem

B. McCollum, P.J. McMullan, A. J. Parkes, E.K. Burke, S. Abdullah

Abstract A two phased approach incorporating the extended great deluge technique is detailed in relation to the Examination Timetabling Problem as described in the 2nd International Timetabling Competition (ITC2007). The approach proves to be both robust and general. Robust in the sense that it is capable of producing six of the best results published in literature so far on the benchmark datasets and general as the technique has produced in the recent past some of the best results on existing course timetabling benchmark datasets. The datasets used as part of this research, introduced during ITC2007, are described and discussed in detail. We present the results of our technique in relation to the competition results and provide a comparison between the outlined method and those of the competition entrants from the international arena, in order to highlight both characteristics of the technique on the datasets used.

B G C McCollum

School of Electronics, Electrical Engineering and Computer Science, Queen's University,
Belfast

E-mail: b.mccollum@qub.ac.uk

P J McMullan

School of Electronics, Electrical Engineering and Computer Science, Queen's University,
Belfast

E-mail: p.p.mcmullan@qub.ac.uk

A J. Parkes

School of Computer Science, University of Nottingham

E-mail: ajp@cs.nott.ac.uk

E K Burke

School of Computer Science, University of Nottingham

E-mail: ekb@cs.nott.ac.uk

Salwani Abdullah

Center for Artificial Intelligence Technology, Universiti Kebangsaan Malaysia, 43600 UKM,
Bangi, Selangor, Malaysia

E-mail: salwani@ftsm.ukm.my

1 Introduction

The automated timetabling of examinations associated with courses within universities is often a complex and time consuming task. From a practical perspective, the quality of the final solution is usually measured firstly in terms of how workable the solution is in terms of resource assignment e.g. availability of students, accessibility of rooms, ordering of examinations, etc. and secondly, on how well a number of institutional defined quality measures are satisfied e.g. how well student exams are spread out throughout the designated examination period [1]. From a research perspective, this measure of quality translates into the satisfaction of what are termed hard and soft constraints and is often achieved through an automatic scheduling process comprising of a two-stage methodology involving the construction and subsequent improvement of a timetabled solution. Construction involves achieving a feasible solution whereby no hard constraints are broken e.g. no students should be expected to take more than one examination at one time. Improvement minimises the violation of desirable constraints termed soft constraints e.g. a student having to sit two examinations in a row. The quality of the final solution is assessed by measuring the outcome after these phases i.e. the degree of satisfaction of hard and soft constraints. Although the problem is similar across universities, various formulations of the problem have appeared within the academic literature over the past fifty years or so. Many search based techniques have been applied to various formulations of the problem and the associated datasets. For a detailed overview of these the reader is referred to [2].

Recently, the 2nd International Timetabling Competition [3] had examination timetabling of one of its three tracks. This track introduced a formulation of the problem which incorporated and brought together a number of real world constraints [4]. This problem formulation adds significantly to previously used models within the research field as many more real world constraints were described and whose cost to overall solution quality were incorporated within the evaluation process. In addition, during the competition, twelve associated data sets were released to the research community. The problem studied here relates to that introduced formulation. Detailed information relating to this exam timetabling formulation, details of the other two tracks, and also overall competition rules can be found at the competition website (www.cs.qub.ac.uk/itc2007). Five finalists from the original submissions were chosen by the competition organisers based on reported results. Importantly, a limit was placed on the computation time allowed for competitor's technique to produce a solution. Competition organisers ran the finalists submitted techniques on a number of previously non-released datasets ('Hidden'), as well as those datasets released as part of the competition ('Early and Late'). Subsequently an ordering was calculated to decide the overall winner based on performance on all the datasets. This can be viewed at <http://www.cs.qub.ac.uk/itc2007/winner/bestexamtrack.htm>.

Here, the organisers of the examination track report the results achieved through the use of an two phase approach incorporating an adaptive construction phase followed by the employment of an extended great deluge technique. The purpose of running the organisers' solvers on the instances and releasing these results is twofold: On one hand, the organisers had to ensure that a feasible solution could be reached for all instances, thus the results are a proof of existence of a feasible solution. On the other hand, from the optimisation point of view, these results form what can be considered a baseline (upper-bound) that can be exploited in future comparisons.

The remainder of this paper is as follows; Section 2 provides the necessary information on the formulation of the examination timetabling problem as introduced as part of ITC2007; Section 3 provides a background to the technique reported as part of this work; Section 4 describes the approaches taken by the five finalists who took part in the competition. It should be noted that all background material described is related to these finalists. This will be updated as further researchers trial their techniques on the formulation and datasets; Section 5 describes the detailed implementation of the extended great deluge along with the results

obtained. Finally the paper is concluded making comment on the effectiveness of the technique studied and potential future research areas.

2 The Examination Timetabling Model

The formulation of the Examination Timetabling Problem introduced as part of ITC2007 is described as ‘post-enrolment’, as the individual examinations each student is taking are known before the timetabling process. This is an important distinction within the area of University Timetabling as the closely related problem, that of course timetabling, can be considered as either post-enrolment or curriculum based. Conflicts between course events can be determined either by common students in the post-enrolment model, or by a hierarchical modular structure within the curriculum based model. Both of these formulations are described as part of ITC2007 [3]. The timetabling process associated with examination timetabling involves placing exams into a number of pre-defined periods within a defined examination session while satisfying a number of hard and soft constraints. A feasible solution is one in which all hard constraints are satisfied. As described in the introduction, the quality of the solution is measured in terms of soft constraints satisfaction. The characteristics which are used to define an individual problem instance include the examinations, students and enrolments, resources such as periods and rooms, hard constraints applied to resources, and general institutional soft constraints. Table 1 lists the main characteristics for each of the examination competition data sets.

Instance	Conflict Density (%)	Exams	Students	Periods	Rooms	Period HC	Room HC
<i>Exam_1</i>	5.05	607	7891	54	7	12	0
<i>Exam_2</i>	1.17	870	12743	40	49	12	2
<i>Exam_3</i>	2.62	934	16439	36	48	170	15
<i>Exam_4</i>	15.0	273	5045	21	1	40	0
<i>Exam_5</i>	0.87	1018	9253	42	3	27	0
<i>Exam_6</i>	6.16	242	7909	16	8	23	0
<i>Exam_7</i>	1.93	1096	14676	80	15	28	0
<i>Exam_8</i>	4.55	598	7718	80	8	20	1
<i>Exam_9</i>	7.84	169	655	25	3	10	0
<i>Exam_10</i>	4.97	214	1577	32	48	58	0
<i>Exam_11</i>	2.62	934	16439	26	40	170	15
<i>Exam_12</i>	18.45	78	1653	12	50	9	7

Table 1 – Problem Benchmark Characteristics

The conflict density is a measurement of the number of conflicts examinations i.e. how tightly the problem is constrained in terms of student enrolments. The initial observation is that the conflict density for most of the data sets is quite low (for the most part around 5% or 6%). This is reflective of the amount of choice allowed to students within the modern curriculum, with a large variation in course or subject choices between each student. The measurable problem ‘size’ (number of exams and students) varies to a certain extent across the set of problems, the largest of which could be argued as either *exam_3/exam_11* or *exam_7* and the smallest as *exam_9* or *exam_12*. The periods and rooms available will also have a measurable effect on the difficulty of achieving feasibility and/or a quality solution. The instances *exam_3* and *exam_11* are practically the same data sets, although *exam_11* has a much smaller set of period resources available with which to timetable. The instances used reflect the ‘real-world’ nature of the data sets which are encountered in actual institutions. In the case of *exam_3* and

exam_11 a common situation arises where the examination session must be shortened to minimize space and staff costs, although all existing constraints must still be adhered to as much as possible.

Information on the structure, length and number of individual periods is also made available. An examination session is made of a number of periods over a specified length of time. This can range from one to two weeks in relation to the data provided. Period lengths, within which a set of examinations or varying duration must be placed, range from one to three hours. A set of rooms and associated capacities are provided.

A feasible timetable is one in which all examinations have been assigned to a period and room so that the following hard constraints are satisfied:

- No student sits more than one examination at the same time;
- The capacity of individual rooms is not exceeded at any time throughout the examination session;
- Period duration restrictions are not violated;
- Period related hard constraints e.g. Exam_A must be placed after Exam_B;
- Room related hard constraints e.g. Exam_A must use Room 101.

A candidate timetable is penalised for each occurrence of the following soft constraints:

- Student has to sit two exams in a row (adjacent on same day);
- Student has to sit two exams in a day;
- Student does not have a specified spread (in terms of periods) of examinations;
- Mixed durations of examinations occur within individual periods;
- Examinations of large class sizes appear later in the examination session;
- Period related soft constraints;
- Room related soft constraints;

As can be seen, these constraints can effectively be split into two groups; those which are resource specific and those which can have a global setting. Resource specific constraints can be set for each period and each room and allows 'control' of how resources can be used in constructing a solution. Global Setting constraints can be set relative to each other. Within the described model, institutions weight these soft constraints differently relative to one another in an attempt to produce a solution which is appropriate for their particular needs. This is defined as the Institutional Model Index. This is a relative weighting of the soft constraints which effectively provides a quality measure of the solution to be built. The Period and Room Hard Constraints will also add to the measurable difficulty of each problem set, although it can be seen that Room Hard Constraints are rarely enforced, and when used, to a limited extent. The amount and type of Period Hard Constraints are reasonably similar across the data sets. However, given the size and amount of exams and enrolments in some of the problem instances compared to resources available, some could be considered more difficult in this regard and possibly more difficult to schedule, both in terms of achieving feasibility and in obtaining a competitive evaluation score.

3 Competition Entrants and Placings

A summary of the competition results is presented in Table 3a in Section 5. However the overall placings are as follows.

Tomáš Müller from Purdue University was the competition winner, having achieved ten out of twelve of the best scores. The algorithm described used a three phased incremental approach [5]. During construction an Iterative Forward Search algorithm was employed in finding an initial feasible solution. During each iteration of the algorithm an examination is chosen and assigned to a room and time. If the assignment causes a hard constraint to be broken then the existing assignment which is causing the problem is unassigned. The process ends when all examinations are assigned a room and time. The algorithm employs both

ordering and assignment heuristics in order to speed up the process. In addition Conflict-based Statistics are used during the iterations in an attempt to avoid repetitive assignments which have previously proven to be detrimental to the developing solution. During the second phase of the algorithm, hill climbing is used to find the local optimal. A neighbourhood is chosen with equal probability from a determined list relating to swapping/changing periods and rooms for randomly chosen examinations. This phase is terminated after a specified number of iterations during which no improvement is experienced. The Great Deluge Algorithm is then engaged in an attempt to improve the solution through widening the search.

Christos Cogos from University of Patras, Greece, was placed second in the competition. His method utilised a GRASP (Greedy Randomized Adaptive Search Procedure) based process combined with other meta-heuristic techniques [6]. The construction phase begins by building five lists of examinations based on various criteria. A tournament based algorithm is used in selecting which exam should be placed in the timetable until all lists are empty. This is carried out iteratively using different starting points in relation to initial time periods. A backtracking strategy, employing a tabu list is employed as required. A Simulated annealing procedure is used in the second phase. In a third phase, integer programming using branch and bound is used to scrutinize and analyse individual periods with the purpose of room changes. Periods are chosen based on an ordering based on overall satisfaction (CSP) of particular soft constraints.

Atusta et al. from Japan used a constraint satisfaction problem solver incorporating tabu search and iterated local search [7,17]. By specifying initial weights, the solver distinguishes between soft and hard constraints and their weights are dynamically controlled during computation to improve performance. The instances were formulated using linear 0-1 inequalities, quadratic 0-1 inequalities, and all-different constraints. The technique proved to be very effective across all three tracks of the competition.

Geoffrey De Smet from Belgium incorporated local search techniques within the Drools solver [8]. Drools is an Open-Source Business Rule Management System (BRMS) (<http://www.jboss.org/drools/>). The developers have provided an integrated environment which allows problem specific semantics to be linked with Domain Specific Languages, graphical editing tools, web based tools and developer productivity tools. The approach described develops the Drools-solver incorporating the Drools rule engine and a local search mechanism. Initially each constraint is written in the Drools Rule Language. Examinations are subsequently ordered on the basis of size and duration. Examinations are assigned to the 'best' position within the timetable chosen by a placing heuristic. This is followed by a local search mechanism which uses three neighbourhoods related to moving and swapping time periods and rooms. The searching continues based on a heuristic which incorporates a tabu based approach.

Nelishia Pillay from the University of KwaZulu-Natal, South Africa came fifth. Inspired by a biological approach, the algorithm mimics cell behaviour [9]. After examinations are ordered heuristically using saturation degree, the examinations are sequentially allocated to the available "cells" within the timetable structure i.e. available times. If more than one time allocation is possible the choice is based on minimum overall solution penalty. If there is a choice once again, allocation is made on a random basis. Rooms are chosen on a best fit heuristic with respect to rooms. Eventually, no cells remain where an examination on the list can be placed without breaking a hard constraint. When this is the case, the already placed examination is moved to a cell which allows minimisation of the overall soft constraint penalty. This is described as cell division. Once this is not possible without the subsequent breaking of hard constraints, a process called cell interaction takes place. This involves a swapping process with the purpose of removing the hard constraint. This process continues until a feasible solution has been reached. The authors liken this to the development of a fully functional organism. Once a feasible solution is reached, improvement is achieved through a process known as cell migration. This involves heuristically swapping the contents of cells that have equal durations.

4 The EGD Approach

The EGD approach has a construction phase followed by improvement. Construction is implemented using the existing adaptive ordering heuristic from [10]; it uses a weighted order list of the examinations to be scheduled based on individual soft penalties and 'difficulty to schedule' penalties. Weightings are increased for each examination based on localised penalties encountered as each are placed, with unscheduled examinations given a much larger increase, based on a formulation involving the maximum general penalty encountered. This latter is an extension to [10] and has been seen in experimentation to achieve improved construction solutions to the standard adaptive construction technique. Once feasibility is achieved, the heuristic continues with the aim of providing an improved solution. It has been found that it is preferable that the construction phase should continue until approximately 9% of the entire process has completed, at which point the improvement phase begins [11].

Figure 1: Extended Great Deluge Algorithm

```

Set the initial solution  $s$  using a construction heuristic;
Calculate initial cost function  $f(s)$ 
Set Initial Boundary Level  $B_0 = f(s)$ 
Set initial decay Rate  $\bullet B$  based on Cooling Parameter
While stopping criteria not met do
    Apply neighbourhood Heuristic  $S^*$  on  $S$ 
    Calculate  $f(s^*)$ 
    If  $f(s^*) \leq f(s)$  or  $(f(s^*) \leq B)$  Then
        Accept  $s = s^*$ 
    Lower Boundary  $B = B - \bullet B$ 
    If no improvement in given time  $T$  Then
        Reset Boundary Level  $B_0 = f(s)$ 
        Set new decay rate  $\bullet B$  based on Secondary
        Cooling Parameter
    
```

Improvement is carried out through implementation of an extended version of the Great Deluge Algorithm. Pseudo code for the algorithm for the Extended Great Deluge is presented in Figure 1. The Great Deluge (also known as Degraded Ceiling) was introduced by Dueck [12] as a faster alternative to Simulated Annealing. It uses a boundary condition rather than a probability measure with which to accept worse solutions i.e. if the penalty function is below a certain value a move in the selected neighbourhood will be automatically accepted, but if it is above it will be rejected. The boundary is initially set slightly higher than the initial solution cost, and reduced gradually through the improvement process. This has already been applied successfully to construction and improvement techniques in timetabling [13,14]. In addition, [15] describes a modification of the Great Deluge algorithm for Multi-criteria decision making. The extended technique employs a reheat mechanism with guided parameters to avoid local optimum and attempt to provide a much wider search of the solution neighbourhood [15]. The aim of this approach is to both improve the speed at which an optimal solution can be found

and at the same time utilise the benefits of this technique in avoiding the trap of local optima. Once again, in order to reduce the amount of time taken, relatively simple neighbourhood moves are employed. In further explanation of the difference, generally, the Great Deluge will terminate when a lack of improvement has been observed for a specified amount of time, as the best solution using a particular neighbourhood has been reached. Rather than terminating, the extended approach employs reheating in order to relax the boundary condition to allow worse moves to be applied to the current solution. Cooling continues and the boundary is reduced at a rate according to the remaining length of the run.

Two basic neighbourhoods are employed within local search, i.e. random moving and swapping individual examinations, while maintaining feasibility and attempting to continually improve the solution or keep the evaluation within a given boundary limit. The application of these simple heuristics allows the algorithm to explore more neighbourhoods efficiently in a limited time environment (such as with a time-limited competition), as computation time is not generally spent on making specific choices for moves or swaps. A full description of the algorithm along with the results achieved in relation to a set of course timetabling problems is given in [16].

The parameters used within the Extended Great Deluge determine the behaviour of the method during improvement, and are listed as follows:

- Initial Boundary
- Decay Rate
- 'Wait for Non-Improvement' parameter
- Post-reheat boundary
- Post-reheat decay rate
- Probability of heuristics employed

The primary parameters used, initial boundary and initial decay rate, dictate how fast the boundary is reduced and ultimately the narrowing condition for accepting worse moves. The approach outlined in this paper uses a Decay Rate proportional to 50% of the entire run. Based on initial iteration timings and the remaining time to produce a final solution within the time limit, the total number of iterations allowed can be calculated in order to determine the appropriate proportions for decay rate. This faster decay will force the algorithm to attempt to reach the optimal solution by, at the very most, half-way through the process. Generally, a continuous lack of improvement will occur before this is reached, at which time the re-heat mechanism is activated.

The 'wait' parameter dictates when to activate the re-heat mechanism due to lack of improvement, specified in terms of percentage or number of total moves in the process. Through experimentation with a number of data set instances a general value for this parameter was established, roughly 5% of the complete run. After reheat the Boundary ceiling is once again set to be greater than the current best evaluation by a similar percentage to that applied in the initial boundary setting. The subsequent decay is set to a 'quicker' rate than with the initial decay, in order to increase the speed of the exploration of neighbouring solutions for improvement. The general setting chosen for the algorithm outlined is set to 25% of the remaining time, with the improvement wait time remaining unchanged. All parameters were established as the most effective as a result of experimentation. The two heuristics used are "Move" (random examination is moved to a random timeslot) and "Swap" (two random examinations swap timeslots), while ensuring that a feasible solution is maintained. Both heuristics may cause a new choice of room allocation if required. There are approximately two "Moves" made for each "Swap".

The choice of parameter values is historical from application to the original course timetabling problems. The ultimate values chosen through experimentation proved highly effective in obtaining best results for these problem data sets, but required further application to a wider range and type of data sets in order to ensure these were not tuned to the specific

data sets used. As we shall see in the next section, the results obtained when applying the technique to the new competition data sets are also highly competitive to current results from the other entrants. It was important to use the same parameter values for these data sets rather than apply new values in order to obtain best results.

5 Experimental Results

As with the competition submissions, the algorithms described here are stochastic, meaning that different runs are generated with different random seeds. During construction of a feasible solution, random weightings may be given for equally ‘difficult to schedule’ examinations. Random choices of examinations, periods and to a certain extent room allocations allow a more diverse search of each current neighbourhood. Statistics calculated from multiple runs are therefore presented in order to provide information on the types of solutions these algorithms are able to produce in general. Firstly for EGD we performed 51 runs on each instance using a different seed in each case and, for each run, calculated the soft cost of the best solution found. From these, we then identified the worst, median, and best solutions that were obtained in the 51 runs, together with the upper and lower quartiles. Note that by using 51 runs here (instead of, say, 50), we are able to obtain the median without the need for interpolation. By using 51 runs rather than the 10 or 11 of the competition we are also able to give reasonable results for the quartiles.

Instance	Best	Q1	Median	Q3	Worst
<i>Exam_1</i>	4633	4750	4799	4852	4889
<i>Exam_2</i>	405	405	425	430	441
<i>Exam_3</i>	9064	9214	9251	9388	9440
<i>Exam_4</i>	15663	15764	15821	15991	16365
<i>Exam_5</i>	3042	3062	3072	3104	3149
<i>Exam_6</i>	25880	25915	25935	26000	26080
<i>Exam_7</i>	4037	4091	4185	4257	4268
<i>Exam_8</i>	7461	7563	7599	7689	7827
<i>Exam_9</i>	1071	1071	1071	1076	1079
<i>Exam_10</i>	14374	14464	14552	14629	14668
<i>Exam_11</i>	29180	29257	29358	29594	29699
<i>Exam_12</i>	5693	5693	5699	5711	5751

Table 2 - Results using the EGD method

Table 2 provides a summary of the results achieved by the EGD algorithm. The first purpose of these results is to give an indication of the variability between runs of the algorithm. The next point to note is that feasibility was gained on all of the data sets. Some of the data sets took a little longer to achieve feasibility during the construction phase than others, but all achieved a feasible solution within one minute of construction. In general, the most difficult solutions proved to be *Exam_1*, *Exam_5* and *Exam_6*, with the improvement getting stuck in local optima more often than the others. As expected, the larger data sets took more computation time per solution generation and evaluation, therefore less iteration time could be spent on these within the imposed competition time limit. *Exam_2* was an interesting case, as the evaluation function based on the soft constraints imposed gave a much smaller value than with any of the others.

The “ITC-2007” block of results in Table 3a provides the best values (highlighted in bold) obtained using the submitted competitor’s techniques using 11 independent runs on each

instances (that is, using a different random seed for those entries that used randomised methods). The results in this block were obtained by the organisers under the competition rules. The result format is a dash indicating that no feasible solution was achieved. Otherwise, the best overall soft constraint violation score is provided.

Solver Instance	(a) ITC-2007 (11 runs per instance)					(b) Post ITC-2007 (51 runs per instance)	
	Muller	Cogos	Atsuta et al	De Smet	Pillay	Muller (ITC2007 code)	EGD
<i>Exam_1</i>	4370	5905	8006	6670	12035	<u>4370</u>	4633
<i>Exam_2</i>	400	1008	3470	623	3074	<u>385</u>	405
<i>Exam_3</i>	10049	13862	18622	-	15917	9378	<u>9064</u>
<i>Exam_4</i>	18141	18674	22559	-	23582	<u>15368</u>	15663
<i>Exam_5</i>	2988	4139	4714	3847	6860	<u>2988</u>	3042
<i>Exam_6</i>	26950	27640	29155	27815	32250	26365	<u>25880</u>
<i>Exam_7</i>	4213	6683	10473	5420	17666	4138	<u>4037</u>
<i>Exam_8</i>	7861	10521	14317	-	16184	7516	<u>7461</u>
<i>Exam_9</i>	1047	1159	1737	1288	2055	<u>1014</u>	1071
<i>Exam_10</i>	16682	-	15085	14778	17724	14555	<u>14374</u>
<i>Exam_11</i>	34129	43888	-	-	40535	31425	<u>29180</u>
<i>Exam_12</i>	5535	-	5264	-	6310	5357	5693

Table 3 - Combined Results, (a) The entries in the first block are results for entrants as obtained in the competition itself, (b) The second block contains the more recent results.

The winner, Tomas Muller, produced 10 of the best results from the top five competitors. Cogos achieved 10 feasible solutions. Atsuta produced 11 feasible solutions and recorded the best result on *Exam_12*. De Smet achieved 7 feasible solutions and the best recorded score for *Exam_10*. Pillay achieved 12 feasible solutions. Note that although Cogos came second, he only achieved 50% feasibility with the hidden data sets *Exam_9* to *Exam_12*, and had the worst result for *Exam_11* compared to Pillay (fifth place) who achieved 100% feasibility and had a better result than Cogos on *Exam_11*. This could suggest that Pillays technique is more effective as a general solution, in terms of feasibility at least, than that employed by Cogos.

Given that Muller was the winner of the competition, Table 3b compares Mullers original submission and the EGD technique on a larger number of runs than the competition. Note that the 51 runs for EGD make it incomparable with the competition results as such. Also, we do emphasise that this should not be taken to reflect badly on the initial entrants. In particular the EGD is produced after the competition and so has had significantly more development time. Furthermore, the parameters within EGD were selected on the basis of access to all 12 instances; in contrast, the competition entrants did not have access to the hidden instances and so might well be argued to be less suitable on those.

Under this comparison Muller and EGD both get 6 best each, and so it seems that EGD is comparable to existing state of the art techniques, and from previous application to other data sets and a different problem domain (course timetabling), can be considered as a generalised technique to solving timetabling problems. It will be interesting to run all techniques at some future point with further hidden data sets, in order to provide a wider test for consistency in the approaches outlined above.

6 Conclusion and Future Work

This paper has described the application of a two-phased approach to solving the Examination Timetabling problem introduced as part of the 2nd International Timetabling Competition. An adaptive heuristic is used to gain feasibility during an initial construction phase. Improvement is achieved through employment of a variant of the Great Deluge (Degraded Ceiling) algorithm. The approach attempts to exploit the inherent advantages with this Extended Great Deluge technique in escaping from local optima while also maintaining a relatively simple set of neighbourhood moves. This is particularly effective here as it reduces the time required for an iterative search process, allowing a balance to be achieved between diversification and intensification within the search strategy. As has been shown, the approach has been successful in achieving a general improvement in solution evaluation as compared to currently published results for the examination timetabling problem, given a common termination criterion.

It is stressed that the results presented here are obtained by examining a greater number of runs than those used during the verification stage of ITC2007. As detailed this was to allow more detailed analysis of the technique. Although, the rules of the competition did not permit the organisers to enter for sound academic reasons, it was felt here for the sake of fairness, the competition time limit should be adhered to in reporting results. In future comparisons, it is proposed various time limits will be used to allow analysis of the effectiveness of techniques when more realistic times are available for the provision of solutions.

In general, this research is part of a much wider analysis of the technique in order to determine whether further improvement can be achieved by modification of the associated parameters and variables used in the process, as outlined in Section 4, while retaining generality for all data sets. It is intended to provide some mechanism for self-adaptation of these parameters based on characteristics of the data sets under consideration as part of this goal. In addition the relationship between the construction and improvement will be investigated to establish rules between the trade-off between the two phases. This would further add to the capability of providing a general algorithm which can ultimately be used for both examination and course timetabling.

References

1. B.McCollum, A Perspective on Bridging the Gap between Research and Practice in University Timetabling (2007), Practice and Theory of Automated Timetabling VI (eds. E.K.Burke and H.Rudova), Lecture Notes in Computer Science Volume 3867, Springer 2007, 3-23.
2. R.Qu, E.K.Burke, B.McCollum, L.G.T.Merlot, S.Y.Lee, A Survey of Search Methodologies and Automated System Development for Examination Timetabling, Accepted for publication by Journal of Scheduling. online publication October 2008, doi: 10.1007/s10951-008-0077-5.
3. B. McCollum, A.Schaerf, B.Paechter, P. McMullan, R.Lewis, A. Parkes, L. Di Gaspero, R.Qu, E. Burke, Setting The Research Agenda in Automated Timetabling: The Second International Timetabling Competition, Accepted for publication to INFORMS Journal of Computing, To be published 2009.
4. B.McCollum, P.McMullan, E.K.Burke, A.J.Parkes, R.Qu, A New Model for Automated Examination Timetabling, Submitted Post PATAT08 special issue of Journal of Scheduling, Available as technical report QUB/IEEE/Tech/ITC2007/Exam/v4.0/17, From http://www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index.htm

5. Muller, T., ITC2007 Solver Description: A Hybrid Approach, Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, 19-22, August 2008.
6. Cogos, C., Alefragis, P., Housos, E., A Multi-Staged Algorithmic Process for the Solution of the Examination Timetabling Problem, Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, 19-22, August 2008.
7. Mitsunori Atsuta, Koji Nonobe, and Toshihide Ibaraki (Japan), ITC2007 Track 1: An Approach using general CSP solver. www.cs.qub.ac.uk/itc2007
8. Geoffrey De Smet: ITC2007 - Examination Track, Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, 19-22, August 2008.
9. Pillay, A Developmental Approach to the Examination timetabling Problem. www.cs.qub.ac.uk/itc2007
10. Burke, E. K., Newall, J. P., (2004), "Solving Examination Timetabling Problems through Adaptation of Heuristic Orderings", Annals of operations Research, 129: 107-134.
11. E. K. Burke, G. Kendal, B. McCollum, P. McMullan (2007) , Constructive versus Improvement Heuristics: An Investigation of Examination Timetabling, 3rd Multidisciplinary International Scheduling Conference: Theory and Applications, 28-31 August 2007, Paris.
12. Dueck, G., (1990), Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing", J. Computational Physics, Vol. 90, 161-175.
13. Burke, E. K., Bykov, Y., Newall, J. P., Petrovic, S., (2003) "A Time-Predefined Approach to Course Timetabling", Yugoslav J. of Operational Research (YUJOR), 13(2): 139-151.
14. Burke, E. K., Bykov, Y., Newall, J. P., Petrovic, S., (2004) "A Time-Predefined Local Search Approach to Exam Timetabling Problems", IIE Transactions, 36(6), 509-528.
15. Petrovic, S., Burke, E. K., (2004), University Timetabling, Handbook of Scheduling: Algorithms, Models and Performance Analysis, CRC Press, Chapter 45.
16. McMullan, P (2007), An Extended Implementation of the Great Deluge Algorithm for Course Timetabling, Lecture Notes in Computer Science, Springer, Vol 4487, 538-545, 2007.
17. K. Nonobe, and T. Ibaraki (2001), An Improved tabu search method for the weighted constraint satisfaction problem, INFOR, Vol. 39, No.2, pp. 131-151, 2001.

Hybridizing Integer Programming Models with an Adaptive Decomposition Approach for Exam Timetabling Problems

Rong Qu • Fang He • Edmund K. Burke

Abstract The idea of decomposition has been successfully applied to solve large combinatorial optimization problems. In timetabling, it has not been widely applied due to one major problem that early assignment may lead to later conflict in solving interrelated sub-problems decomposed from the original problem. In our previous work, timetabling problems were adaptively decomposed into *difficult set* and *easy set* of exams of different importance by the information gathered from previous iterations during the problem solving. The approach obtained promising results and showed that small set of difficult exams contributed to a much larger portion of the total cost of the solution constructed. An interesting issue would then be to investigate the effect of constructing complete solutions based on the optimal solution of this difficult set sub-problem.

In this paper, we first present an IP formulation for solving the difficult set sub-problem. To have a tighter initial formulation, a well-known valid inequality named Clique Inequality is utilized. We then examined the combinatorial properties of the problem to introduce a new family of cutting planes. These showed to be helpful on obtaining optimal solution of a gap $< 10\%$ for the sub-problem, based on which the final solution is constructed. Promising results have been obtained on several benchmark exam timetabling problems in the literature.

1 Introduction

Exam timetabling is an important administrative management problem in universities and educational institutions, and has been extensively investigated in the last four decades [27]. Educational systems are different worldwide, but the basic task of exam timetabling is similar. It consists of assigning exams of registered students to a limited number of time periods so that no conflict occurs (e.g. no student has to take more than one exam at a time). Another common constraint is to spread students' exams as far as possible to allow them enough revision time. Constraints are usually problem specific, and are given different priorities (weightings) in different institutions. The quality of timetables is usually evaluated by how much these constraints are satisfied. Timetabling problems belong to the class of NP-hard and complexity issues were also addressed [13]. Due to their importance in practice and challenges in research, exam timetabling problems have been widely investigated in both operational research and artificial intelligence community. For a comprehensive review of methodologies in exam timetabling, we refer a recent survey in [27].

Due to the large size and complex features of exam timetabling problems, IP models have not been widely investigated in the research. The only work on exam timetabling we are aware

Rong Qu

Automated Scheduling, Optimization and Planning (ASAP) Group, School of Computer Science,
University of Nottingham, Nottingham, NG8 1BB, UK
E-mail: rxq@cs.nott.ac.uk

Fang He

Automated Scheduling, Optimization and Planning (ASAP) Group, School of Computer Science,
University of Nottingham, Nottingham, NG8 1BB, UK

Edmund K. Burke

Automated Scheduling, Optimization and Planning (ASAP) Group, School of Computer Science,
University of Nottingham, Nottingham, NG8 1BB, UK

of is in [19], where an IP model was built to solve a specific problem with less number of constraints. Some IP formulations have been studied in course timetabling, where Branch-and-Bound [14] and Branch-and-Cut [2] were employed. Different cutting planes have been introduced to accelerate the Branch-and-Bound algorithm.

Very often real-world exam timetabling problems are very large and difficult. Heuristic approaches have been employed to solve the problem efficiently. Simple graph based heuristics have been investigated [7, 11], and are still being employed and adapted within hybridized methods in the current research [6, 8, 26]. Meta-heuristics have become the state-of-the-art [27]. Local search based techniques such as Simulated Annealing [4, 18, 23], Tabu search [15, 22, 28] have been widely studied. Evolutionary algorithms have also been investigated throughout of the years [5, 16, 24], hybridized with different techniques including multi-objective techniques [12]. In recent research, hybrid approaches [9, 18] have also been successfully applied in timetabling, integrating different techniques such as constraint based techniques and graph based heuristics.

The basic idea of decomposition is to “divide and conquer”, as optimal solutions of smaller sub-problems may be much easier to obtain by using relatively simple approaches or even exact method. However, the task of decomposition is challenging and problem specific and little work exist during the years in the complex timetabling problems. Carter and Johnson [10] extended the scope of clique to include a larger subset of quasi-clique (dense sub-graphs) as the basis of building good complete solutions. Their results showed that the quasi-clique initialization which identifies sufficiently dense sub-graphs in exam timetabling problems can significantly improve the quality of complete solutions. Qu and Burke [25] adaptively decomposed the problem into two sets of exams (*difficult set* and *easy set*) of different difficulty level. Difficulty of exams in specific problems is adaptively adjusted by the information gathered from previous iterations of the problem solving. The approach obtained promising results compared against the best results from a number of approaches in the literature at that time. In particular, it was showed that the adaptive difficult set is usually of small size, but contributes to the major portion of the total cost of the constructed solution. A simple random greedy technique was used to find a good enough solution to the difficult set sub-problem in [25], and thus the effectiveness of the approach was not warranted.

In this paper, we carried out an in-depth analysis on these adaptively identified difficult sets. The effect of the optimal solution for the crucial sub-problems within the complete solution is studied. The difficult set sub-problems are formulated into a set packing problem with side constraints, and solved by employing an exact IP model to obtain the optimal solution (within a gap $< 10\%$). To tight the initial formulation, we studied the structure of exam timetabling problems and introduced new valid inequalities based on the specific combinatorial properties of the problem. Computational results show that the quality of solutions for the difficult set sub-problems contributes to high quality final complete solutions constructed.

This paper is organized as follows: Section 2 presents details of the benchmark exam timetabling problems we are solving and overviews our hybrid adaptive decomposition approach. Section 3 elaborates the IP formulation of the problem and the problem specific cutting planes introduced in the IP model. Section 4 reports the computational results and analysis, followed by the conclusion and future work in Section 5.

2 The Benchmark Exam Timetabling Problems and the Hybrid Adaptive Decomposition Approach

2.1 The benchmark exam timetabling problems

The benchmark exam timetabling problems we investigate in this paper is the University of Toronto problem, firstly introduced by Carter et al. [11] and publicly available at

<ftp://ftp.mie.utoronto.ca/pub/carter/testprob>. Over the years they were widely employed as testbeds in exam timetabling research. However, there has been an issue with different instances circulating under the same name. To encourage future scientific comparisons, the issues were clarified and different versions of the instances in the dataset were renamed as version *I*, version *II* and version *IIc* (see more details in [27]). We use version *I* of the dataset and present the characteristics of these problems in Table 1. More details of the problems and an API evaluation function are provided at <http://www.asap.cs.nott.ac.uk/resources/data.shtml>.

Table 1 Characteristics of the benchmark exam timetabling problems in [11], also see [27]

Problem instance	No. of Exams	No. of Students	No. of Time Periods	Conflict Density
car83 I	190	1125	24	0.27
hec92 I	81	2823	18	0.42
lse91	381	2726	18	0.06
sta83 I	139	611	13	0.14
tre92	261	4360	23	0.18
yor83 I	181	941	21	0.29

In the problem dataset, to indicate the density of conflicting exams (i.e. those exams with common students thus cannot be scheduled in the same time period), a *Conflict Matrix* C was defined where each element $c_{ij} = 1$ if exam i conflicts with exam j , or $c_{ij} = 0$ otherwise. The *Conflict Density* represents the ratio between the number of elements c_{ij} of value “1” to the total number of elements in the conflict matrix. A high conflict density indicates the problem is highly constrained thus may be difficult to solve. The objective is to minimize the average cost per student, calculated by an evaluation function defined as follows. For students sitting two exams s time periods apart, a proximity cost w_s is assigned, i.e. $w_1 = 16$, $w_2 = 8$, $w_3 = 4$, $w_4 = 2$, $w_5 = 1$ and $w_s = 0$ for $s > 5$. The aim is to space out the conflicting exams within a limited number of time periods to allow more revision time for students.

2.2 The hybrid adaptive decomposition approach

In our previous adaptive decomposition approach [25], an initial ordering of the exams is first obtained by using the Saturation Degree graph heuristic. Exams are ordered increasingly by the number of remaining feasible time periods for them during the solution construction. Earlier exams with less number of feasible time periods in the ordering can be seen as more difficult and are scheduled to the timetable first. In some cases, the ordering may need to be randomly adjusted until a feasible solution can be obtained. Based on this initial ordering, this list of exams is then decomposed adaptively into two subsets (named the *difficult set* and the *easy set*). The pseudo-code presented in **Algorithm 1** outlines the adaptive decomposition process to identify the *difficult set*.

The approach is an iterative process where one solution is built at each iteration by scheduling the exams in *difficult set* and *easy set* in the ordering at that iteration. The exams in *difficult set* are adaptively adjusted at each iteration by using information derived from the solution construction in previous iterations i.e. the aim is to identify troublesome exams, which can be dealt with higher priority in future iterations of solution construction. Problematic exams which cannot be scheduled in the previous iteration are added into *difficult set*, and the quality of the previous solution is used to adjust the size of *difficult set*.

After *difficult set* is identified by the above process, ordering of exams in the *easy set* is then randomly adjusted, while *difficult set* is fixed, to find better solutions. It was observed that detecting the problematic exams in the *difficult set* is crucial in building high quality solutions, i.e. better solutions after identifying the *difficult set* usually led to better final solutions after adjusting the *easy set*. However, adjusting exams in *easy set* also contributed to building good solutions, i.e. solutions after the *difficult set* is identified still can be greatly improved in some cases. We refer to the work in [25] for more analysis on the identified *difficult set*.

Algorithm 1: Adaptive Decomposition by Identifying the Difficult Set

```

Build a feasible initial ordering of exams based on Saturation Degree heuristic
Initial size of difficult set  $S_d$  = number of exams / 2 //  $S_d \in [1, \text{number of exams} - 1]$ 
MaxNoIterations = 10,000; iteration = 0
while iteration < MaxNoIterations do
    easy set =  $\{e_{S_d+1}, e_{S_d+2}, \dots, e_e\}$ 
    reorder the exams in the difficult set  $\{e_1, e_2, \dots, e_{S_d}\}$ 
    construct a solution by using ordered exams in both difficult set and easy set
    if a feasible solution or an improved solution is obtained
         $S_d = S_d + 1$  // include more potential exams in the difficult set
    else
        move forward the difficult exam causing the infeasibility in the ordering
        set the size of the difficult set to where the difficult exam was
        store the difficult set and its size  $S_{best}$  if the best solution is obtained
    iteration = iteration + 1
end while

```

At each iteration, the exams are scheduled one by one according to their ordering in the *difficult set* and *easy set* into the time period leading to the least cost. In the case of ties, i.e. assigning the exam to two time periods leads to the same cost, the exam is randomly scheduled to one of the time periods. It was observed in our previous work that assigning the exam to the earliest or randomly selected time period does not produce significantly different timetables.

In this work, the difficult set sub-problem detected by the above adaptive decomposition is input to the IP model (see Section 3) to obtain the optimal solution, based on which the complete solution is constructed by scheduling the exams in the *easy set*. A simple steepest descent method is then applied to quickly improve the complete solution built based on the optimal solution of the difficult set sub-problem. The aim here is to demonstrate and analyze the effectiveness of building solutions based on optimal solutions for crucial sub-problems. Different advanced meta-heuristics could of course be employed to further improve the complete solution constructed and provide better results given more computational time.

3 The IP formulation for the sub-problem

3.1 The IP formulation

We define the following notations to be used in our model:

Set:

E , set of exams

S , set of students registered to different exams in E

T , set of time periods for scheduling all exams

Data:

GS_e , the group of students enrolled to exam e , $e \in E$

GE_s , the group of exams to which student s enrolled, $s \in S$

Variables:

Two core binary variables and five additional counting variables are defined:

$X_{et} = 1$ if exam e is scheduled in time period t , 0 otherwise.

$Y_{st} = 1$ if student s is scheduled an exam in time period t , 0 otherwise.

C_1 is the number of violations in conflicting type 1, i.e. two conflicting exams of a student are scheduled one time period away.

$C_1 = \sum_s \sum_t C_{1st}$, $C_{1st} = 1$ if student s has two exams one time period away starting from t , 0 otherwise.

$C_2 - C_5$ are defined in the same way, i.e. the number of violations in conflicting types 2-5, i.e. two conflicting exams scheduled two-five time periods away

Different types of conflicting are given different importance, thus a weight is defined as:

$w_1 = 16$, $w_2 = 8$, $w_3 = 4$, $w_4 = 2$, $w_5 = 1$ for two exams scheduled one, two, three, four and five time periods away, respectively

Objective function:

$$\text{Min } \sum w_i C_i$$

$$\text{s.t. } \sum_t X_{et} \leq 1, \forall e \in E; \quad (1)$$

$$\sum_{e \in GS_e} X_{et} \leq 1, \forall s \in S, \forall t \in T; \quad (2)$$

$$X_{et} \leq Y_{st}, \forall e \in E, \forall t \in T, \forall s \in GS_e; \quad (3)$$

$$Y_{st} + Y_{st+1} - C_{1st} \leq 1; \quad (4)$$

$$Y_{st} + Y_{st+2} - C_{2st} \leq 1; \quad (5)$$

$$Y_{st} + Y_{st+3} - C_{3st} \leq 1; \quad (6)$$

$$Y_{st} + Y_{st+4} - C_{4st} \leq 1; \quad (7)$$

$$Y_{st} + Y_{st+5} - C_{5st} \leq 1; \quad (8)$$

The objective is to space out conflicting exams within the limited number of time periods T . Constraint (1) requires that one exam can only be scheduled into exactly one time period; Constraint (2) defines that student cannot take two exams at the same time; Constraint (3) forces the following condition to define the relationship between exams and students, if $X_{et} = 1$, then $\forall s \in GS_e$, $Y_{st} = 1$, otherwise $Y_{st} = 0$. Constraints (4)-(8) are used to count the number of violations for students, i.e. for student s and time period t , $C_{1st} = 1$ if two exams are scheduled one time period away (i.e. $Y_{st} = 1$ and $Y_{st+1} = 1$), otherwise $C_{1st} = 0$.

3.2 Clique Inequality cutting planes

To derive valid inequalities for exam timetabling problems, we examine the relation between our problems and the Set Packing problem. The aim is to exploit the connection between these two problems from the polyhedral description of the Set Packing polytope [3], whose structure has been extensively investigated.

The set packing problem belongs to combinatorial optimization problems. Given a finite set S , the problem is to find the maximum number of pairwise disjoint subsets in S , i.e. no two subsets intersect. The problem can be defined as the following 0-1 integer linear programming formulation:

$$\text{Max } \{CX, AX \leq e^T, x_i = 0 \text{ or } 1 \text{ for } i = 1, \dots, n\} \quad (9)$$

where A is an $m \times n$ matrix with entries in $\{0, 1\}$; X is a set of n variables and C is a cost vector.

Constraints (1) and (2) in the above formulation take the form of a Set Packing problem (9), and define a relaxation of our exam timetabling problem without considering other constraints (3)-(8). We therefore use the Set Packing relaxation as a logical framework to derive a well-known family of cutting planes, namely Clique Inequalities [17, 21], which present to be very effective in tightening the formulation for our exam timetabling problems.

Let $G(V, E)$ be the intersection graph associated with the Set Packing relaxation and let $P(G)$ denote the Set Packing polytope, i.e., the convex hull of the incidence vectors of the

stable sets in $G(V, E)$. A maximal clique is a maximal complete sub-graph of G . Let $K \subset V$ be a subset of nodes inducing a clique. It is well known that the Clique Inequality

$$\sum_{k \in K} x_k \leq 1 \quad (10)$$

is valid for $P(G)$ and it is facet-inducing if K is a maximal clique [2]. This Clique Inequality is implemented within our above IP formulation in CPLEX 10.0.

3.3 Problem specific cutting planes

We introduce two classes of valid inequalities to obtain a further tighter formulation for exam timetabling problems.

1. Exam/time period cuts: if exams e and e' involve the same students, then $X_{et} + X_{e't} \leq 1$.

Proposition 1: $X_{et} + X_{e't} \leq 1, \forall e \in E, \forall t \in T$, if $GS_e = GS_{e'}$, hold for exam timetabling problems.

Proof. If exams e and e' involve the same students s , then exams e and e' must be included in the list GE_s . By constraint (2), we have $X_{et} + X_{e't} \leq 1$. $\square$

2. Implied bound cuts: for the cost counting constraints (4)-(8), we can obtain the bound of C_i according to the number of exams each student takes.

Proposition 2: we denote n_1 as the number of time periods; n_2 as the number of exams student s take; $n_2 \leq n_1$; we then have the implied bound of C_i as

$$\max\{0, n_2 - (\lfloor n_1 / 2 \rfloor + 1)\} \leq C_1 \leq n_2 - 1;$$

$$\max\{0, n_2 - (\lfloor n_1 / 3 \rfloor + 1)\} \leq C_2 \leq n_2 - 1;$$

$$\max\{0, n_2 - (\lfloor n_1 / 4 \rfloor + 1)\} \leq C_3 \leq n_2 - 1;$$

$$\max\{0, n_2 - (\lfloor n_1 / 5 \rfloor + 1)\} \leq C_4 \leq n_2 - 1;$$

$$\max\{0, n_2 - (\lfloor n_1 / 6 \rfloor + 1)\} \leq C_5 \leq n_2 - 1;$$

Proof. The upper bound of C_i indicates the worst case scenario (consecutive exams to the student). For example, Figure 1 shows the maximum violations of type 1 for a student. The lower bound of C_i is the best case scenario (exams are spread as much as possible). Figure 2 shows the minimum violations of type 1 for a student. The upper and lower bounds of types 2-5 can be obtained in the same way. $\square$

Figure 1. The maximum violations of type 1 a student may take ("X" represents the student has an exam in the time period. $n_1 = 13, n_2 = 9, C_1 \leq 8$)

X	X	X	X	X	X	X	X	X				

Figure 2. The minimum violations of type 1 the student take ("X" represents student has exam at the time period. $n_1 = 13, n_2 = 9, 2 \leq C_1$)

X	X	X	X	X		X		X		X		X

4 Computational results

We report our analysis and computational experience on applying the IP formulation with cutting planes in the adaptive decomposition approach for solving some representative instances in the benchmark exam timetabling problems given in Section 2.1. Computational experiments have been carried out on a personal computer with Intel Core 1.86GHz processor

and 1.97GB RAM. We have used the callable library ILOG CPLEX as LP solver using LP-based Branch-and-Bound technique for solving integer linear programs. CPLEX provides functionalities to manage the tree search. ILOG CPLEX Barrier was used as the LP solver, and cutting planes (valid Clique inequalities) were generated throughout the search tree [20]. The problem specific cutting planes were only added statically at root node of the Branch-and-Cut tree. All other CPLEX settings are set to default (i.e. CPLEX automatically chooses the heuristic strategy, node and variable selection strategies, etc).

4.1 Analysis of the adaptive difficult set

In [10], a phase-based approach was employed where the first phase is to allocate a large set of mutually conflicting exams (i.e. cliques in the associated graph). A solution to a maximum clique, arbitrarily chosen among a number of cliques, was used as the initial solution. It was shown that clique initialization in exam timetabling can be potentially improved by identifying at the outset larger sets of exams which form sufficiently dense sub-graphs to be *almost* a clique. This procedure is static and carried out offline for each specific problem in hand.

In our approach, *difficult set* which consists of troublesome exams is identified adaptively by the information obtained from online problem solving in the previous iterations. We first exam the adaptive *difficult set* and compare it with the maximum clique for the specific problem instance. Our aim of decomposition is twofold: on the one hand, we try to identify as many troublesome exams as possible. Not only the exams in maximum cliques, but also the exams that form almost clique are considered; on the other hand, we only need real difficult exams which cause trouble in timetables in our difficult set. Including non-crucial easy exams in *difficult set* could significantly increase the computational time in searching for the optimal solution by using the IP model for the difficult set sub-problem, and not contributing to the high quality final complete solution constructed.

Table 2 presents the sizes of the original problem, their maximum clique and *difficult set*. We can see that the size of *difficult set* is larger than the size of the maximum clique so it includes more troublesome exams. Also the adaptive *difficult set* forms almost clique (see an example in Table 3). Note that the *difficult set* is adaptively obtained rather than by setting an accurate threshold value of the density.

Table 2. Sizes of the original problem, its maximum clique and the difficult set identified

		ear83 I	hec92 I	lse91	sta83 I	tre92	yor83 I
Original problem	no. students	1125	2823	2726	611	4360	941
	no. time periods	24	18	18	13	23	21
	no. exams	190	81	381	139	261	181
	max-clique size	21	17	17	13	20	18
Difficult set	no. students	999	1592	1717	269	1677	788
	no. exams	54	46	54	13	28	53

Table 3 presents the elements and the density of the *difficult set* for an example instance *sta83 I*. It is interesting to see that the adaptive difficult set identifies the clique, i.e. it includes all elements in a clique. Also the adaptive difficult set and the quasi-clique reported in [10] have a large overlap of elements (with only one element different), indicating that more troublesome exams identified form sufficiently dense sub-graphs. The quality of the final complete solution built based on the clique, quasi-clique and adaptive difficult set is also presented, showing that the complete solution built based on difficult set is the best compared against those built based on clique and quasi-clique sets. This indicates that cliques of higher density, which are statically measured, may not form a good base for final complete solutions. Adaptive difficult set identified online potentially contribute to higher quality final solutions constructed.

Table 3. Properties of different sub-problems: set size, set elements, density of the set and the best complete solution for the original problem based on the sub-problems.

Sets	Size	Set elements	Density	Final best solution
Clique	12	{4,18,27,39,59,72,86,107,121,133,136,139}	1.00	167.84
Quasi-clique	13	{4,18,27,39,59,72,86,107,121,122,133,136,139}	0.93	157.51
Difficult set	13	{4,18,27,39,59,72,79,86,107,121,133,136,139}	0.86	157.12

4.2 Analysis of the solutions to the difficult set sub-problem

The difficult set sub-problems are solved by our IP model with Clique inequalities and problem specific cutting planes to reduce the number of branches searched. Cutting planes can be added either at root node or other nodes. In our approach, we only add cutting planes at the root node. Table 4 presents more details of the model with and without the added inequalities and cutting planes. The second column gives the size of the original problem formulation. The third column gives the number of Cliques Inequalities generated by CPLEX. Note that while the cuts added by CPLEX reduce the running time for most problems, they may occasionally have the opposite effect. If CPLEX added too many cuts at the root node and the objective value does not change significantly, we turn off the added cuts. The forth column is the reduced problem size after pre-processing and aggregating. The “Barrier LP” column indicates the lower bound of the LP relaxation. The last column is the optimal integer solution with the specific gap.

Table 4. LP relaxations of the problem formulation, with Clique Inequalities added automatically and implied bounds added statically at the root node. Columns 2-6 present the problem size of original formulation (row*column); number of cliques; reduced problem size after reductions in CPLEX (row*column, non-zeros); LP relaxation result and the best IP result (within the gap of 10%).

Sub-problem	Original LP size	Clique table members	Reduced LP size		Barrier LP	IP results (gap<10%)
ear83 I	215976*130435	19631	103823*104730	360534	0.00	15.4985
hec92 I	200414*144137	16830	83280*83226	270372	0.00	6.94221
lse91	220754*156549	20104	96004*95088	321299	0.00	7.03902
sta83 I	29394*18072	3119	15069*15213	50202	3.13	16.0335
tre92	243043*188046	11413	60613*61160	188425	0.00	2.36017
yor83 I	141126*88133	12212	64322*65382	219267	15.50	16.3185

The quality of the solutions to the sub-problem plays an important role in the final complete solutions. Table 5 gives the solutions built based on the sub-problems of different quality (by setting different gaps to the optimal). A steepest descent method is also applied to quickly improve the complete solution. It shows that the better the solution of the IP sub-problem (see the column “Initial”), the better the final complete solution would be (see the column “Improved”). $\Delta\%$ presents the improvement of the solution quality to the next smaller gap, i.e. improvement from gap = 70% to gap = 60%, improvement from gap = 60% to gap = 10%. It shows that the smaller the gap of the optimal solution to the sub-problem, the better the final complete solution. However, it is also observed by experiments that a smaller gap of optimal solution to the sub problem, say gap < 1%, dramatically increase the computational time.

An issue to be investigated in our future work is to set a good balance between the quality of the sub-solution and the final complete solution, and the computational time. Due to inter-related constraints between the sub-problem and the other exams in the problem, the optimal solution of gap 0 to the sub-problem may not be in the complete global optimal solution. It is therefore not necessary to spend a large computational time to obtain zero-gap optimal solutions for the sub-problems, i.e. there is no need to find the optimal solution of gap 0 for the sub-problems identified.

Table 5 Complete solutions from optimal solution of different gaps for the difficult set sub-problem. $\Delta\%$ = (result of gap 1 - result of gap 2) / result of gap 2; "Initial" indicates the initial solution built based on the IP sub-solution; "Improved" indicates the improved solution after the simple steepest descent method. Results are average values from 10 runs.

		IP (gap = 70%)		IP (gap = 60%)		IP (gap = 10%±5)	
		Results	$\Delta\%$	Results	$\Delta\%$	Results	$\Delta\%$
ear83 I	Initial	47.56	--	46.28	2.7	43.14	7.2
	Improved	40.30	--	38.53	4.5	37.92	1.6
hec92 I	Initial	13.50	--	13.26	1.8	12.30	7.8
	Improved	12.28	--	11.91	3.1	11.74	1.4
lse91	Initial	12.76	--	11.85	7.6	11.12	6.5
	Improved	11.89	--	11.48	3.5	11.11	3.3
sta83 I	Initial	168.01	--	167.06	0.5	160.78	3.9
	Improved	167.84	--	160.61	4.5	157.50	1.9
tre92	Initial	11.12	--	10.60	4.9	9.92	6.8
	Improved	9.92	--	9.55	3.9	9.38	1.8
yor83 I	Initial	52.30	--	50.76	3.0	45.76	10.9
	Improved	45.48	--	43.70	4.0	42.62	2.5

4.3 Analysis of the complete solutions

To give more insights of the performance of our IP based approach, we compared statistically the performance of our IP approach with the previous adaptive decomposition approach in [25]. Table 6 presents the statistical distributions of the results from these two approaches.

Table 6. Results from our previous work [25] and our IP approach. The best average, best result and the lowest standard deviation (s.d.) are given in *italics* and **bold**.

		ear83 I	hec92 I	lse91	sta83 I	tre92	yor83 I
[25]	Average	37.84	12.09	11.31	157.60	8.98	43.27
	Best	36.15	11.38	10.85	157.21	8.79	42.2
	s.d.	1.40	0.43	0.38	0.28	0.15	1.03
Our IP approach	Average	37.72	11.74	11.11	157.5	9.38	42.69
	Best	37.30	11.48	11.07	157.12	9.24	41.97
	s.d.	0.68	0.18	0.04	0.13	0.11	0.51

It can be seen that our IP based approach obtained better average results on 5 out of 6 problems being considered. It also has smaller standard deviation for all the problems, meaning that its performance is more stable. This is because in [25] a simple random greedy technique is used to build the solution to the sub-problem, and thus the effectiveness of the approach was not warranted. In our IP based approach, solutions to the crucial sub-problems are solved optimally.

Table 7 compares our approach with a number of other approaches in the literature. The best results are highlighted. The first three approaches are all constructive based approaches where improvement meta-heuristics are not the main approach to generate the solutions for the problems. Our approach obtained similar results. The last three approaches generate at least one of the best results in the current literature for the problem instances being considered. Our IP based approach obtained the best result in the literature for one instance *sta83 I*. It should be noted that so far the best results for the benchmark problem instances are generated from a range of state-of-the-art approaches in the literature. No single approach can be seen as the best for solving all problem instances.

Table 7. Best results from our IP based approach, compared with other approaches in the literature.

	ear83 I	hec92 I	lse91	sta83 I	tre92	yor83 I
graph heuristics [11]	36.4	10.8	10.5	161.5	9.6	41.7
adaptive ordering [6]	36.16- 38.55	11.61- 12.82	10.96- 12.53	161.91- 170.53	8.79- 8.96	40.77- 42.97
adaptive decomposition [25]	36.15	11.38	10.05	157.21	8.79	42.2
Our IP based approach	37.30	11.48	11.07	157.12	9.24	41.97
large neighborhood search [1]	34.87	10.28	10.24	159.2	8.13	36.11
hybrid local search [9]	29.3	9.2	9.6	158.2	9.2	36.2

4.4 Discussions and future work

It would be interesting that, in our future work, we extend the study in several directions. Firstly, the solutions to the sub-problem are not of optimal with gap 0%. It would be interesting to investigate the trade-off between obtaining sub-problem solutions closer to real optimal and the computational time required. Due to the interrelations between exams in the *difficult set* and the *easy set* in the problem, the optimal sub-solution may not remain the same in the global complete solution. The task is not necessarily to search for the real optimal solution for the difficult set sub-problem, but rather to find a high quality solution which may form a good basis for a complete solution which is closer to the global optimal solution. Secondly, as the solutions of sub-problems are fixed during the construction the complete solution, the final complete solution is restricted within some specific regions of the search space. It would be interesting to investigate how much should this restriction be relaxed so that potential global optimal solutions are not excluded from the search of the approach. Thirdly, it worth further investigation to examine the size of the *difficult set* with regard to the troublesomeness of the exams included. Due to the interrelated constraints between exams in the timetabling problem, the static measurements such as the degree of difficult exams, the maximum cliques and dense enough quasi-cliques are not enough to provide good evaluations on scheduling the exams into the timetable. Our future work would include designing more robust decomposition methods to include real troublesome exams in the crucial sub-problems. Finally, more efficient Branch-and-Cut procedures will be investigated to obtain optimal solutions with closer gap for larger size sub-problems.

5 Conclusions

Due to the large computational time of exam methods, in the literature there is little work on IP for exam timetabling problems. In this paper, we developed and evaluated an IP model for solving sub-problems of exam timetabling problems identified by an adaptive decomposition approach. The difficult sets which are adaptively decomposed from the original large problems are solved optimally by the IP model we proposed. To get a tighter formulation for the problem, we not only induce Clique Inequalities deduced from Set Packing problems, but also introduce problem specific cutting planes. Better average complete solutions can be built based on the solution of the difficult set sub-problem compared with the previous adaptive decomposition approach. Closer investigations on the identified difficult set sub-problem with cliques and quasi-cliques of higher density, and complete solutions built based on optimal solutions of different gaps for the sub-problems are also carried out.

The quality of the complete solution lies on the size and elements of the sub-problem identified and their solutions from the IP model. Due to that the IP formulation of real-world timetabling problems have a huge number of variables and constraints, in the future work we will investigate more problem specific properties to obtain more efficient formulations, Branch-and-Cut procedures and alternative LP solution techniques. Larger crucial sub-

problems can be solved and robust techniques can be employed to further improve the adaptive decomposition approach.

References

1. Abdullah S., Ahmadi S., Burke E.K. and Dror M. Investigating Ahuja–Orlins large neighbourhood search for examination timetabling. *OR Spectrum*, 29(2), 351-372, 2007
2. Avella P. and Vasil'ev I. A computational study of a cutting plane algorithm for university course timetabling. *Journal of Scheduling* 8, 497-514, 2005
3. Borndorfer R. and Weismantel R. Set packing relaxations of some integer programs. *Mathematical Programming*, 88, 425-450, 2000
4. Burke E.K., Y. Bykov, J.P. Newall, and S. Petrovic. A time-predefined local search approach to exam timetabling problems. *IIE Transactions*, 36(6), 509-528, 2004
5. Burke E.K. and Newall J.P. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1), 63-74, 1999
6. Burke E.K. and Newall J.P. Solving examination timetabling problems through adaptation of heuristic orderings. *Annals of OR*, 129, 107-134, 2004
7. Burke E. K., Newall, J. P. and Weare, R. F. A simple heuristically guided search for the timetable problem. In Proceedings of the international ICSC symposium on engineering of intelligent systems (EIS98), 574–579, 1998.
8. Burke E. K., Hart E., Kendall G., Newall J., Ross P. and Schulenburg S. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover & G. Kochenberger (eds.), *Handbook of Meta-heuristics*, 457-474, 2003
9. Caramia M., Dell'Olmo P. and Italiano G.F. Novel local search-based approaches to university examination timetabling. *INFORMS Journal of Computing*, 20(1), 86-99, 2008
10. Carter M.W. and Johnson D.G. Extended clique initialization in examination timetabling. *Journal of Operational Research Society*, 52, 538-544, 2001
11. Carter M.W., Laporte G. and Lee S.Y. Examination timetabling: Algorithmic strategies and applications. *Journal of Operational Research Society*, 47(3), 373-383, 1996
12. Cote P., Wong T. and Sabouri R. Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem. In: Burke E.K. and Trick M. (eds.) *Practice and Theory of Automated Timetabling V: Selected Papers from the 5th International Conference*. Lecture Notes in Computer Science, vol. 3616. 151-168, 2005
13. de Werra D. The combinatorics of the timetabling. *European Journal of Operational Research*, 96, 504-513, 1997
14. Dascalki S.T. Birbas, and E. Housos. An integer programming formulation for a case study in university timetabling. *European Journal of Operational Research*, 153, 117-135, 2004
15. Di Gaspero L. Recolour, shake and kick: A recipe for the examination timetabling problem. In: Burke E.K. and De Causmaecker P. (eds.). *Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling*, 21st-23rd August 2002. KaHo St.-Lieven, Gent, Belgium. 404-407, 2002
16. Erben W. A grouping genetic algorithm for graph colouring and exam timetabling. In: Burke E.K. and Erben W. (eds). *Practice and Theory of Automated Timetabling III: Selected Papers from the 3rd International Conference*. Lecture Notes in Computer Science, vol. 2079. 132-156, 2001
17. Hoffman K.L. and Padberg M. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6), 657-682, 1993
18. Merlot L.T.G., N. Boland, B.D. Hughes and P.J. Stuckey. A hybrid algorithm for the examination timetabling problem. In: Burke E.K. and De Causmaecker P. (eds). *Practice and Theory of Automated Timetabling IV: Selected Papers from the 4th International Conference*. Lecture Notes in Computer Science, vol. 2740. 207-231, 2003
19. MirHassani S.A. Improving paper spread in examination timetables using integer programming, *Applied Mathematics and Computation* 179, 702-706, 2006
20. Nemhauser G.L. and Wolsey L.A. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1999
21. Padberg M.W. On the facial structure of the set packing polyhedra, *Mathematical Programming*, 5, 199-215, 1973
22. Paquete L. and T. Stutzle. Empirical analysis of tabu search for the lexicographic optimization of the examination timetabling problem. In: Burke E.K. and De Causmaecker P. (eds), *Proceedings of the*

- 4th International Conference on Practice and Theory of Automated Timetabling. 21st-23rd August 2002. KaHo St.-Lieven, Gent, Belgium. 413-420, 2002
23. Thompson J. and K. Dowsland. A robust simulated annealing based examination timetabling system. *Computers & Operations Research*, 25, 637-648, 1998
 24. Ross P., Hart E. and Come D. Some observations about GA-based exam timetabling. In: Burke E.K. and Carter M.W. (eds.) *Practice and Theory of Automated Timetabling II: Selected Papers from the 2nd International Conference*. Lecture Notes in Computer Science, vol. 1408. 115-129, 1998
 25. Qu R. and Burke E.K. Adaptive decomposition and construction for examination timetabling problems. *Proceedings of the 3rd Multidisciplinary International Scheduling: Theory and Applications*, 418-425, 2007
 26. Qu R., Burke E.K. and McCollum B. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. To appear at *European Journal of Operational Research*, 2009, doi:10.1016/j.ejor.2008.10.001
 27. Qu R., Burke E. K., McCollum B., Merlot L.T.G. and Lee S.Y. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1), 55-89, 2009
 28. White G.M., Xie B.S. and Zonjic S. Using tabu search with longer-term memory and relaxation to create examination timetables. *European Journal of Operational Research*, 153(16), 80-91, 2004

Evolving Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem

Nelishia Pillay

Abstract This paper presents a genetic programming (GP) hyper-heuristic approach that optimizes a search space of functions to assess the difficulty of allocating an examination during the timetable construction process. Each function is a heuristic combination of low-level construction heuristics combined by logical operators. The approach is tested on a set of five benchmark problems of varying difficulty to evaluate its ability to generalize. The GP hyper-heuristic approach was found to generalize well over the five problems and performed comparably to other hyper-heuristic approaches combining low-level construction heuristics.

1 Introduction

Much of the research in the domain of examination timetabling has been aimed at developing methodologies that produce the best quality timetables for a problem [21]. A more recent direction of research in this field, namely, hyper-heuristic methods, focuses on creating algorithms that generalize well over a set of problems. A common approach taken by hyper-heuristic methods involves combining low-level heuristics. This paper presents a genetic programming hyper-heuristic approach for evolving heuristic combinations in the form of a function comprised of a combination of low-level constructive heuristics and logical operators. This approach was tested on a set of five benchmark problems of differing difficulty. The approach was found to generalize well and produced results comparable to other hyper-heuristic approaches combining low-level heuristics.

The following section provides an overview examination timetabling. Section 3 examines the use of hyper-heuristic approaches in the domain of examination timetabling. A genetic programming approach for inducing “hierarchical” combinations of low-level construction heuristics is proposed in section 4. The methodology employed for evaluating this approach is described in section 5. Section 6 discusses the performance of the GP hyper-heuristic approach on the set of five benchmark problems. The results of the study and future extensions of this work are summarized in section 7.

Nelishia Pillay
School of Computer Science,
University of KwaZulu-Natal,
KwaZulu-Natal, South Africa
E-mail: pillayn32@ukzn.ac.za

2 Examination Timetabling

The examination timetabling problem is a well-researched domain [21]. The problem requires a set of examinations to be allocated to a given set of examination periods. The timetable generated must meet the *hard constraints* of the problem. The hard constraints are those requirements that must be satisfied by a timetable in order for it to be operable, e.g. no student must be scheduled to write more than one examination during the same period.

A timetable meeting the hard constraints of the problem is called a *feasible* timetable. Each examination timetabling problem also has a set of *soft constraints*. Soft constraints are constraints that we would like the timetable to meet, but which may not always be possible, e.g. examinations must be well-spaced over the examination session for all students. Thus, we attempt to minimize the soft constraint cost, i.e. the number of soft constraints violated. Room capacities are not taken into consideration in the uncapacitated version of the problem. The capacitated version of the problem has an additional hard constraint requiring room capacities to be satisfied for each period.

Research in the area of examination timetabling was initiated by Carter et al. [12] in 1996 and since then numerous methodologies have been applied to this problem. A set of 13 benchmark problems proposed by Carter et al. [12] have been used to compare the performance of the different methodologies in solving the examination timetabling problem. A later set of benchmarks have also been made available as part of the international timetabling competition [16]. This is a more constrained set of problems than the Carter benchmark set.

The study presented in this paper focuses on the Carter benchmark set. A number of methodologies have been cited in the literature as producing the best quality timetables for one or more problems in the benchmark set. The sequential construction method employed by Caramia et al. [11] has produced the best quality timetables for 5 of the problems. This methodology uses a greedy scheduler to firstly obtain a feasible timetable. A penalty decreaser and penalty trader are then used to further improve the quality of the timetable. The algorithm developed by Burke et al. [7] combining the use of variable neighbourhood search and a genetic algorithm has also produced a best quality timetable. Burke et al. [9] introduce the concept of a “late acceptance strategy”. In this approach the hill-climbing algorithm implemented compares a candidate solution with the best solution obtained a few iterations back instead of the current best solution. This algorithm has produced the best quality timetables for four of the benchmark problems. The results obtained by these methodologies are presented in section 6.

Instead of developing methods that produce the best quality timetable for one or more of a set of benchmark problems, research in the domain of examination timetabling is moving towards developing approaches that generalize well over a set of problems. This has led to the creation of hyper-heuristic algorithms. Examination timetables are generally constructed by sorting the examinations according to their difficulty and allocating examinations, in order, to the minimum penalty timetable slot ([12], [21]). A low-level heuristic measures the difficulty of examinations. A hyper-heuristic algorithm is used to determine which low-level heuristic or combination of heuristics to use in deciding on which examination should be allocated next. In addition to constructive heuristics, hyper-heuristic approaches can also generate or combine improvement, perturbative, or hill-climber ([5], [10], [13] and [14]) heuristics. The study presented in this paper is limited to constructive heuristics and previous work focusing on construction heuristics are described in the following section.

3 Hyper-Heuristics and Examination Timetabling

One of two approaches is generally taken by hyper-heuristic algorithms. The first adapts an existing heuristic or determines which single low-level heuristic to use. The second approach combines low-level heuristics in some way. This is achieved by applying a function

to two or more heuristics or searching a space of heuristic combinations. The study presented in this paper employs the second approach and thus this section summarizes previous work evaluating hyper-heuristic algorithms taking this approach in the domain of examination timetabling. Furthermore, all these algorithms have been tested on the same version of the Carter benchmark problems used in this study.

Qu et al. [19] implemented a variable neighbourhood search (VNS) to optimize a search space of heuristic combinations consisting of two or more low-level graph colouring heuristics. Each heuristic in the heuristic combination produced by the VNS is used to schedule two examinations.

Earlier research conducted by Burke et al. [6] applied a Tabu search to the space of heuristic combinations comprised of the saturation and largest degree low-level heuristics. The algorithm was tested on four of the Carter benchmark problems. This work was extended in later research conducted by Burke et al. [8] which employed a Tabu search to explore a search space of heuristic combinations comprised of two or more low-level graph colouring heuristics. Each heuristic is used to allocate two examinations. A further extension of this work used the Tabu search algorithm to produce heuristic combinations that construct feasible timetables [20]. These combinations are then further analysed to determine the most effective distribution pattern of low-level heuristics, for example a particular heuristic may produce better solutions when applied at the beginning of the construction process, i.e. it occurs at the beginning or the end of the combination string. The analyses revealed that the best approach would be to start off with the heuristic combination consisting only of the saturation degree and hybridizing the largest weighted degree heuristic into the combination during the timetable construction process.

Asmuni et al. [1] used a fuzzy logic approach to combine two of three low-level graph colouring heuristics. The fuzzy logic algorithm combines the two low-level heuristics into a single value which is used to estimate the difficulty of allocating an exam. The examinations are sorted according to this fuzzy value and scheduled accordingly. In later work [3], Asmuni et al. [3] consider more combination pairs of the three low-level heuristics. In another study Asmuni et al. [2] combine three low-level heuristics in the same way and tune the fuzzy rules instead of keeping them fixed to produce better quality timetables.

Pillay et al. [17] employ a genetic programming algorithm to optimize a search space of heuristic combinations. Each combination is essentially a sequential list of low-level heuristics. The number of low-level heuristics comprising each combination is randomly chosen to be within the range of a preset maximum. An initial population of heuristic combinations is randomly created and the mutation and crossover operators are used to create successive generations. Tournament selection is used to choose the parents that these operators are applied to.

Pillay et al. [18] propose an alternative approach to combining low-level heuristics. Instead of combining and applying low-level heuristics sequentially, heuristics are combined hierarchically and applied simultaneously. Each combination consists of two or more primary heuristics, with one of the primary heuristics denoted as a priority heuristic to deal with conflicts. The combination may also contain a secondary heuristic which is used to break ties. The heuristics are applied simultaneously by combining the heuristics with logical operators. Four such heuristic combinations are created and tested on the Carter benchmark set.

Section 6 compares the performance of the genetic programming hyper-heuristic approach presented in this paper to the results obtained by the algorithms described in this section.

4 The Genetic Programming Hyper-Heuristic Approach

The study presented in this paper takes a similar approach to that presented in [18] in that low-level heuristics are combined hierarchically and applied simultaneously instead of sequentially.

Genetic programming (GP) is used to explore a space of such combinations. Genetic programming is an evolutionary algorithm based on Darwin's theory of evolution [15]. GP

iteratively refines a program space rather than a solution space. The optimal program found by a GP system is then implemented to find a solution. Genetic programming has been applied successfully to various domains ([4] and [15]).

The GP system implemented in this study is strongly-typed and employs the generational control model. An overview of the algorithm is depicted in Figure 1. An initial population is created by randomly constructing heuristic combinations. This initial population is then iteratively refined through the processes of evaluation, selection and recreation. These processes are described in the sections that follow.

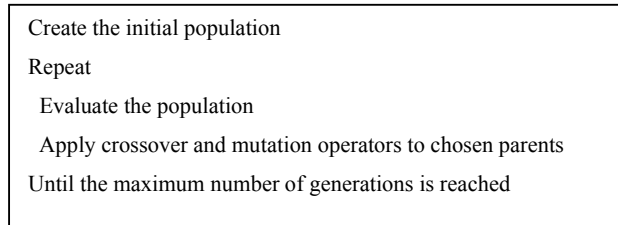


Figure 1. Genetic programming algorithm

4.1 Population Representation and Generation

Each element of the population is represented as a parse tree. An example is illustrated in Figure 2.

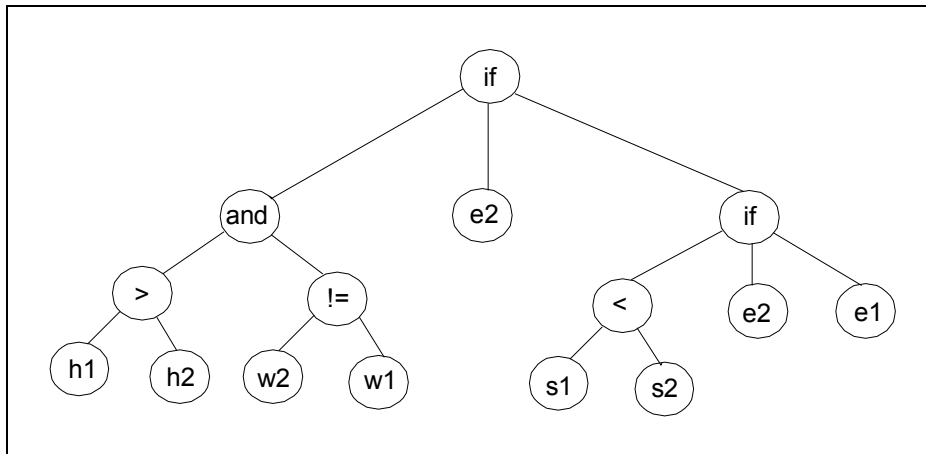


Figure 2. An example of an element of the population

A depth limit is set on the overall tree, and subtrees for the operators. This is to prevent the growth of redundant code. The root node of each individual is the *if* operator. This operator takes three arguments. The first child represents a condition. This node can be an *and* operator or one of the arithmetic logic operators, greater than (>), less than (<), equal to (==) and not equal (!=). These operators were chosen as the combinations producing good results in [18] were composed of these operators. The *and* operator has two children. Each child node can be another *and* operator, if the maximum depth of the subtree has not been reached, or an arithmetic logical operator. The arguments of the arithmetic logical operators are pairs of one of the following construction low-level heuristics:

- Largest degree (d) - The degree of an examination is the number of clashes the examination is involved in.
- Largest enrollment (e) - This refers to the class size, i.e. number of enrollments for an examination.
- Largest weighted degree (w) - This is the number of students involved in clashes for the examination.
- Saturation degree (s) - The saturation degree of an examination is the number of options, i.e. feasible periods an examination can be allocated to, given the current state of the timetable.
- Highest cost (h) - Is the sum of the soft constraint cost of scheduling the examination in each feasible timeslot, given the current state of the timetable.

The first three heuristics are referred to as static heuristics as their values can be calculated prior to the construction process while the values of the latter heuristics change according to the state of a timetable and are called dynamic heuristics.

A heuristic is randomly chosen to be the child of an arithmetic logic operator. The first child is randomly selected to represent the heuristic of the first or second examination being compared, and the second child represents the heuristic of the remaining examination. For example, in Figure 2 the highest cost heuristic is the chosen heuristic for the children of the $>$ operator with the first child chosen to represent the highest cost of the first examination and the second child the highest cost of the second examination.

The second and third child of the *if* operator can be another *if* operator, if the depth limit of the subtree has not been reached, or an examination node. An examination node is denoted by e_1 or e_2 representing the first or second examination respectively. Thus, the output of an individual is e_1 or e_2 indicating which of the two exams should be given priority to be scheduled.

An initial population of m combinations is created, where m is a genetic parameter, i.e. its value is problem dependant.

4.2 Fitness evaluation and selection

Each individual is evaluated by using it to construct a timetable. Consider the three examinations in Table 1 and their corresponding saturation degree, s , and weighted degree, w , heuristic values:

Table 1. Exam example

Exam	s value	w value
E1	4	5
E2	2	20
E3	7	20

If the individual in Figure 3 is used to order the examinations during timetable construction, the exams will allocated in the following order: E2, E3, E1.

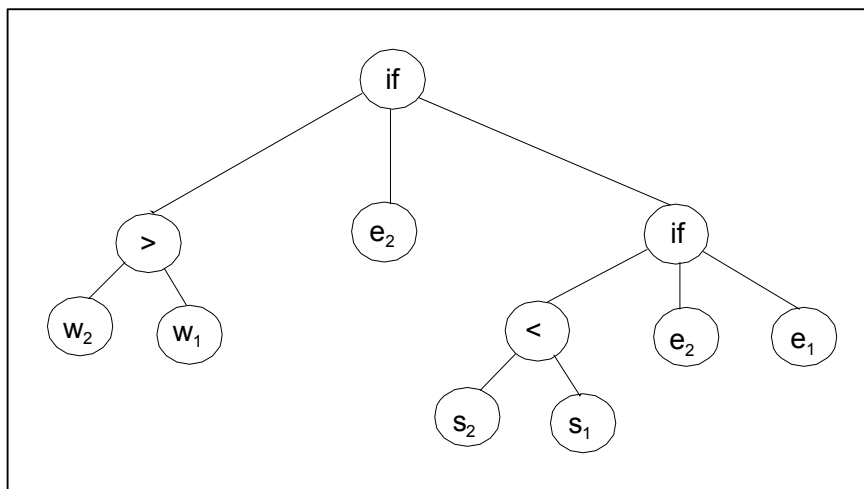


Figure 3. Another example of an individual

The fitness of an individual is a function of the hard constraint and soft constraint cost of the timetable constructed using the individual to order examinations during the timetable construction process. Equation (1) is used for this purpose:

$$\text{fitness} = (\text{hard constraint cost of the timetable} + 1) * (\text{soft constraint cost of the timetable}) \quad (1)$$

The tournament selection method is used by the genetic operators to select parents for the next generation. This selection method randomly chooses s individuals. These individuals form the *tournament*. The individual with the best fitness is the winner of the tournament and is returned as a parent. Selection is with replacement, thus an individual maybe chosen more than once as a parent. The tournament size is a genetic parameter.

4.3 Genetic operators

Two genetic operators are used to create offspring for the next generation, namely, mutation and crossover. The mutation operator randomly chooses a mutation point in the copy of the parent. The subtree rooted at the mutation point is replaced. The crossover operator randomly selects a crossover point in the copy of the first parent. A crossover point of the same type, e.g. an *if* node, an *exam* node or a *heuristic* node, is randomly chosen in the copy of the second parent. The subtrees rooted at the crossover points are swapped. Trials runs revealed that the crossover operator was more effective if the fitter of the two offspring was returned instead of both offspring as is standard practice.

Both the mutation and crossover operators are structure-preserving, i.e. all the nodes in the offspring have children of the correct type. If a heuristic node is replaced its sibling is replaced as well. For example, suppose that the node s_1 is replaced by h_1 , then its corresponding sibling is replaced by h_2 .

In order to stifle intron growth and reduce bloating [4], a limit is set on the size of an offspring in terms of the number of nodes the offspring is comprised of. This value is a genetic parameter. Both mutation and crossover are repeated until an offspring of the permitted size is generated.

The application rates of the mutation and crossover operators are genetic parameters.

5 Experimental Setup

The main aim of the study presented in this paper is to test the effectiveness of exploring a search space of hierarchically combined constructive heuristics, i.e. low-level heuristics combined by logical operators, rather than sequential heuristic combinations such as *shhw*. The approach was tested on the five Carter benchmark problems [21], of differing difficulty, listed in Table 2. The density of the clash matrix is the ratio of the number of examinations involved in clashes and the total number of examinations. This serves as a measure of the difficulty of the problem. The hard constraint for all five problems is that no student must be scheduled to sit for more than one examination during the same period. The soft constraint for the Carter benchmark set is that the examinations must be well spread for all students. The formula in equation (2) is used to calculate the soft constraint cost of the timetable.

$$\frac{\sum w(|e_i - e_j|) N_{ij}}{S} \quad (2)$$

where:

- 1) $|e_i - e_j|$ is the distance between the periods of each pair of examinations (e_i, e_j) with common students.
- 2) N_{ij} is the number of students common to both examinations.

- 3) S is the total number of students
- 4) $w(1) = 16$, $w(2) = 8$, $w(3) = 4$, $w(4) = 2$ and $w(5) = 1$, i.e. the smaller the distance between periods the higher the weight allocated. Note for $n > 5$, $w(n) = 0$

Table 2. Five Carter benchmarks

Problem	Institution	Periods	No. of Exams	No. of Students	Density of Conflict Matrix
ear-f-83 I	Earl Haig Collegiate Institute, Toronto	24	190	1125	0.27
hec-s-92 I	Ecole des Hautes Etudes Commerciales, Montreal	18	81	2823	0.42
sta-f-83 I	St Andrew's Junior High School, Toronto	13	139	611	0.14
ute-s-92	Faculty of Engineering, University of Toronto	10	184	2749	0.08
yor-f-83 I	York Mills Collegiate Institute, Toronto	21	181	941	0.29

The values of the genetic parameters are listed in Table 3. These values were obtained empirically by performing trial runs.

Table 3: Genetic parameter values

Parameter	Value
Number of generations	50
Population size	500
If-subtree depth limit	3
Operator subtree depth limit	2
Tournament size	4
Crossover rate	0.3
Mutation rate	0.7
Maximum offspring size	100

The algorithm was implemented in Java using JDK 1.6.0 and simulations were run on an Intel Core 2 Duo processor with Windows XP. Ten runs, each using a different random number generator seed, were performed for each of the five benchmarks.

6 Results and Discussion

A heuristic combination that produces a feasible timetable was found for all five data sets. Figure 4 depicts the heuristic combination that has produced the best soft constraint cost for the *hec-s-92I* data set. Note that the redundant code has been removed. The saturation degree heuristic has priority. Examinations with the lowest saturation degree are scheduled first. If an examination has an equivalent or higher saturation degree its largest weighted degree is examined. The next part of the evolved function is not typical of a human created function. If the largest weighted degree of an examination is higher and it has an equivalent highest cost value it is scheduled next. The best soft constraint cost, the average soft constraint cost and the average runtime for the five problems are listed in Table 4.

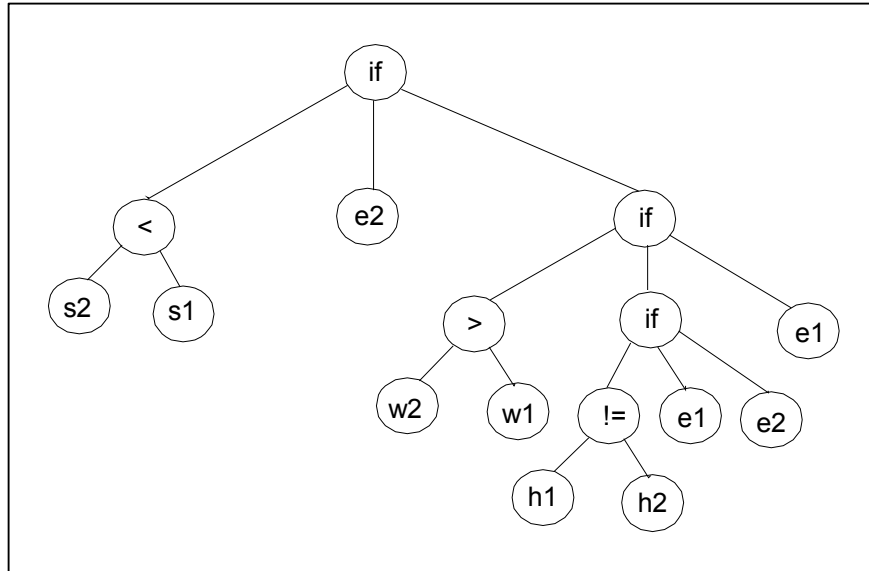


Figure 4. A heuristic combination for the hec-s-92I data set

Table 4. Soft constraint costs and runtime

Data Set	Best Cost	Average Cost	Average Runtime
ear-f-83 I	37.39	37.85	16 hrs
hec-s-92 I	11.43	11.67	50 mins
sta-f-83 I	158.38	158.69	5 hrs
ute-s-92	27.31	28.02	8 hrs
yor-f-83 I	39.96	40.58	14 hrs

The main objective of a hyper-heuristic algorithm is to generalize well over a set of problems rather than produce the best quality timetable for one or two of the problems. Hence, assessing the performance of the GP approach presented in this paper by a direct comparison with the quality of timetables produced by other hyper-heuristic approaches for the five problems will not be meaningful. Instead the performance over all five problems is evaluated and compared. This is achieved by calculating the distance the soft constraint cost is from the best reported in the literature for each problem and summing these values to get an idea of how each hyper-heuristic approach performed over all five problems. These five problems were chosen so as to represent problems of differing difficulty as indicated by the density of the conflict matrix for each problem.

The best results cited in the literature for each of the five problems are listed in Table 5. These include the sequential construction method implemented by Caramia et al. [11], the variable neighbourhood search incorporating the use of genetic algorithms developed by Burke et al. [7] and the hill-climbing with a late acceptance strategy used by Burke et al. [9]. These algorithms are described in section 2.

Table 5. Best results cited for the Carter benchmarks

Data Set	DA	Caramia et al. [11]	Burke et al. [7]	Burke et al. [9]
ear-f-83 I	35.09	29.3	32.8	32.65
hec-s-92 I	11.08	9.2	10.0	10.06
sta-f-83 I	157.28	158.2	156.9	157.03
ute-s-92	26.5	24.4	24.8	24.79
yor-f-83 I	39.4	36.2	34.9	34.78

Table 6 lists the soft constraint cost for each of the data sets for the GP approach presented in this paper and for the following hyper-heuristic approaches (an overview of these approaches have been provided in section 3):

- (1) The variable neighbourhood search implemented by Qu et al. [19]
- (2) The fuzzy logic expert algorithm developed by Asmuni et al. [1]
- (3) The fuzzy logic expert algorithm with fine tuning (Asmuni et al. [2])
- (4) The extended fuzzy logic algorithm with additional models tested by Asmuni et al. [3]
- (5) The tabu search employed by Burke et al. [8]
- (6) The adaptive automated construction approach used by Qu et al. [20]
- (7) The evolutionary algorithm tested by Pillay et al. [17]
- (8) The heuristic combinations evaluated by Pillay et al. [18]

These approaches were chosen for comparison as they take the same approach as the GP algorithm presented in this study, namely, either search a space of heuristic combinations or combine low-level heuristics by means of a function. In all cases the hard constraint cost is zero, i.e. feasible timetables were found for all five problems by all of the hyper-heuristic approaches. Table 7 tabulates the distance from the best cited results for each of the five problems and sum of these values for the GP approach and the approaches listed above.

Table 6. Soft constraint costs for the different hyper-heuristic approaches

Problem	GP	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
ear-f-83 I	37.39	37.29	37.02	36.64	37.02	38.19	35.56	36.94	36.86
hec-s-92 I	11.43	12.23	11.78	11.6	11.78	12.72	11.62	11.55	11.85
sta-f-83 I	158.38	158.8	160.42	160.79	160.42	158.19	158.88	158.22	158.33
ute-s-92	27.31	29.68	27.78	27.55	28.07	31.65	28	26.65	28.88
yor-f-83 I	39.96	43	40.66	39.79	39.8	40.13	40.71	41.57	40.74

Table 7. Comparison of the performance of hyper-heuristic approaches over the five problems

Problem	GP	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
ear-f-83 I	8.09	7.99	7.72	7.34	7.72	8.89	6.26	7.64	7.56
hec-s-92 I	2.23	3.03	2.58	2.4	2.58	3.52	2.42	2.35	2.65
sta-f-83 I	1.48	1.9	3.52	3.89	3.52	1.29	1.98	1.32	1.43
ute-s-92	2.91	5.28	3.38	3.15	3.67	7.25	3.6	2.25	4.48
yor-f-83 I	5.18	8.22	5.88	5.01	5.02	5.35	5.93	6.79	5.96
Total	19.89	26.42	23.08	21.79	22.51	26.3	20.19	20.35	22.08

It is evident from Table 6 and Table 7 that the performance of the GP hyper-heuristic algorithm is comparable to other hyper-heuristic approaches. Furthermore, this algorithm generalizes well over all five problems with the lowest total distance from the best results cited in the literature for each of the problems.

7 Conclusion and Future Work

The study presented in this paper implements a genetic programming approach to evolve functions that calculate the difficulty of scheduling an examination during the timetable construction process. These functions combine low-level construction heuristics and apply them simultaneously. The hyper-heuristic algorithm was tested on a set of five benchmark problems of differing difficulty to assess its ability to generalize. The heuristic combinations generated by the GP algorithm produced feasible timetables for all five problems. The quality of the timetables produced by the GP-based hyper-heuristic approach was comparable to other hyper-heuristic approaches and was found to generalize well over the five problems.

Given the success of the approach with the five problems, future work will include applying the algorithm to additional problems of differing difficulty and characteristics. This will include extending the algorithm for evaluation with the data sets from the international timetabling problem. Future extensions of the algorithm will also investigate the effect of using additional logical and arithmetic logical operators in the heuristic combinations and testing a more structured syntax for hierarchical heuristic combinations, similar to that proposed in [18].

Acknowledgements The author would like to thank the reviewers for their helpful comments.

References

1. Asmuni H., Burke E. K., and Garibaldi J. M., Fuzzy Multiple Ordering Criteria for Examination Timetabling, in Burke E.K., Trick M. (eds.), selected Papers from the 5th International Conference on the Theory and Practice of Automated Timetabling (PATAT 2004)- The Theory and Practice of Automated Timetabling V, Lecture Notes in Computer Science, 3616, 147–160 (2005)
2. Asmuni H., Burke E.K., Garibaldi J. M., and McCollum B., Determining Rules in Fuzzy Multiple Heuristic Orderings for Constructing Examination Timetables, Proceedings of the 3rd Multidisciplinary International Scheduling: Theory and Applications Conference (MISTA 2007), 59-66 (2007)
3. Asmuni H., Burke E.K., Garibaldi J. M., McCollum B., and Parkes A. J., An Investigation of Fuzzy Multiple Heuristic Orderings in the Construction of University Examination Timetables, Computers and Operations Research, Elsevier, 36(4), 981-1001 (2009)
4. Banzhaf W., Nordin P., Keller R.E., Francone F.D., Genetic Programming - An Introduction - On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann Publishers, Inc. (1998)
5. Bilgin B., Ozcan E., and Korkmaz E.E., An Experimental Study on Hyper-Heuristics and Exam Timetabling, in: Burke EK, Rudova H (eds.), Practice and Theory of Automated Timetabling VI: Selected Papers from the 6th International Conference, PATAT 2006, Lecture Notes in Computer Science, 3867, 394 – 412 (2007)
6. Burke E. K. , Dror M. , Petrovic S., and Qu R, Hybrid Graph Heuristics with a Hyper-Heuristic Approach to Exam Timetabling Problems, in Golden B.L., Raghavan S., Wasil E.A. (eds.), The Next Wave in Computing, Optimization, and Decision Technologies – Conference Volume of the 9th Informatics Computing Society Conference, 79 -91 (2005)
7. Burke, E. K., Eckersley, A., McCollum, B., Petrovic, B., and Qu, R., Hybrid Variable Neighborhood Approaches to University Exam Timetabling. Accepted for publication in the European Journal of Operational Research (EJOR). (2009)
8. Burke E.K., McCollum B., Meisels A., Petrovic S., and Qu R, A Graph-Based Hyper-Heuristic for Educational Timetabling Problems, European Journal of Operational Research (EJOR), 176, 177 – 192 (2007).
9. Burke E. K., and Bykov Y., A Late Acceptance Strategy in Hill-Climbing for Examination Timetabling Problems in the Proceedings of the conference on the Practice and Theory of Automated Timetabling (PATAT 2008), (2008).

10. Burke E. K., Misir M., Ochoa G., and Ozcan E., Learning Heuristic Selection for Hyperheuristics for Examination Timetabling, in the Proceedings of the conference on the Practice and Theory of Automated Timetabling (PATAT 2008), (2008)
11. Caramia, M., Dell Olmo, P., Italiano, G. F., Novel Local-Search-Based Approaches to University Examination Timetabling, *INFORMS Journal of Computing*, 20(1), 86-99, (2008)
12. Carter M. W., Laporte G., Lee S.Y., Examination Timetabling: Algorithmic Strategies and Applications, *The Journal of the Operational Research Society*, 47(3), 373-383 (1996)
13. Ersoy E., Ozcan E., and Uyar S., Memetic Algorithms and Hill-Climbers, In: Baptiste P, Kendall G, Kordon AM, Sourd F (eds.), *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications Conference (MISTA 2007)*, 159-166 (2007)
14. Kendall G., and Hussin N.M., An Investigation of a Tabu Search Based on Hyper-Heuristics for Examination Timetabling, in Kendall G., Burke E.K., Petrovic S. (eds), *Proceedings of the 2nd Multidisciplinary Scheduling: Theory and Applications Conference (MISTA 2005)*, 309-328 (2005)
15. Koza J. R., *Genetic Programming I: On the Programming of Computers by Means of Natural Selection*. MIT Press (1992)
16. McCollum B., McMullan P., Paechter B., Lewis R., Schaerf A., Di Gapsaro L., Parkes A. J., Qu R., and Burke E.K., Setting the research agenda in automated timetabling: the second international timetabling competition. Submitted to *INFORMS*. Retrieved June 28, 2008, from <http://www.cs.qub.ac.uk/itc2007/winner/finalorder.htm> (2008)
17. Pillay N., and Banzhaf W., A Genetic Programming Approach to the Generation of Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem, in Neves et al. (eds.), *Progress in Artificial Intelligence, Lecture Notes in Artificial Intelligence*, 4874, 223-234 (2007)
18. Pillay N., and Banzhaf W., A Study of Heuristic Combinations for Hyper-Heuristic Systems for the Uncapacitated Examination Timetabling Problem, to appear in the *European Journal of Operational Research(EJOR)*, doi:10.1016/j.ejor.2008.07.023 (2008)
19. Qu R., Burke E.K., Hybrid Neighbourhood HyperHeuristics for Exam Timetabling Problems, in *Proceedings of the MIC2005: The Sixth Metaheuristics International Conference*, Vienna, Austria (2005)
20. Qu R., Burke E. K., and McCollum B., Adaptive Automated Construction of Hybrid Heuristics for Exam Timetabling and Graph Colouring Problems, to appear in the *European Journal of Operational Research (EJOR)*, doi:10.1016/j.ejor.2008.10.001 (2008).
21. Qu R., Burke E. K., McCollum B., Merlot L.T.G., and Lee S.Y., A Survey of Search Methodologies and Automated System Development for Examination Timetabling, to appear in the *Journal of Scheduling*, doi: 10.1007/s10951-008-0077-5 (2008).

Parallel Machine Problems with Equal Processing Times

Svetlana A. Kravchenko · Frank Werner

Abstract The basic scheduling problem we are dealing with is the following. There are n jobs, each requiring an identical execution time. All jobs have to be processed on a set of parallel machines. Preemptions can be either allowed or forbidden. The aim is to construct a feasible schedule such that a given criterion is minimized. For a couple of problems of this type, recently the complexity status has been solved and several interesting results have been presented. In this paper, we survey existing approaches for the problem class considered.

1 Introduction

We consider the following basic scheduling problem. There are n jobs, denoted by $J_1, J_2, \dots, J_n$, each requiring an identical execution time p . With each job $J_j, j = 1, \dots, n$, there are associated a release time r_j , a due date d_j , a deadline D_j and a weight w_j . All jobs have to be processed on a set of m identical machines. Each machine can process only one job at a time. During the processing of the jobs, preemptions can be allowed or forbidden. In the first case, the processing of any job once started has to be completed without any interruptions. In the second case, the processing of any operation may be interrupted and resumed later, possibly on another machine. The objective is to construct a feasible schedule so as to minimize a given criterion. In this paper, we discuss several solution approaches for the problem considered and expose problems with an open complexity status. For the interested readers, we can recommend in addition the collections of results from [8] and [14].

United Institute of Informatics Problems
Minsk, 220012 Belarus
E-mail: kravch@newman.bas-net.by

Fakultät für Mathematik, Otto-von-Guericke-Universität
Magdeburg, 39106 Germany
E-mail: frank.werner@mathematik.uni-magdeburg.de

2 Problems without preemptions

Whereas the overwhelming majority of scheduling problems appears to be NP-hard, problems with equal processing time jobs form a remarkable case which is still open for most problems. Intuitively, such problems look polynomially solvable.

Due to the possibility to enumerate the possible places of the jobs in an optimal schedule, the model reminds an assignment problem. Therefore, one can intuitively suppose the existence of a polynomial algorithm for any monotonic criterion. Nevertheless, since prime conflicts are caused by overlapping intervals most of the problems have an open complexity status.

A special case is the model with $p = 1$. For most criteria, this model can be solved by a network flow algorithm. However, this approach cannot be applied to the general model with arbitrary p since, in the case of $p = 1$, the main conflicts among overlapping places for processing jobs disappear. We refer the reader to [3] for a survey.

2.1 Classical criteria

If the processing times can be arbitrary, problem $1|r_j, D_j| -$ is unary NP-hard by a polynomial reduction from the 3-partition problem, see [26]. In [17], an $O(n \log n)$ algorithm has been proposed for problem $1|r_j, p_j = p, D_j, prec| C_{max}$. More precisely, the authors consider a problem with unit processing times and arbitrary rational release dates and deadlines. However, by an appropriate scaling one can show that these two models are equivalent. They show that it is possible to modify the release times and deadlines so as to reflect the partial order (i.e., for the problem considered the constraint $prec$ is irrelevant) by assigning $r_j := \max\{r_j\} \cup \{r_i + 1 \mid T_i \prec T_j\}$ and $d_j := \min\{d_j\} \cup \{d_i - 1 \mid T_j \prec T_i\}$. After such an assignment condition, $T_i \prec T_j$ implies $r_i < r_j$ and $d_i < d_j$. By straightforward interchange arguments, one can show that any schedule for problem $1 \mid r_j, p_j = p, D_j \mid C_{max}$ can be transformed into a schedule for problem $1 \mid r_j, p_j = p, D_j, prec \mid C_{max}$ without changing the C_{max} -value. To solve problem $1 \mid r_j, p_j = p, D_j \mid C_{max}$, the concept of forbidden regions (intervals) has been proposed (see [17]), i.e., open intervals where no job can start. The set of all forbidden regions is formed iteratively. The main observation used is the following. Assume that one knows the set of forbidden regions in the interval $[r_i, D_j]$, and let $J_1, \dots, J_k$ be the set of jobs with release times and deadlines from the interval $[r_i, D_j]$. Now we set $r_1 = \dots = r_k = r_i$ and $D_1 = \dots = D_k = D_j$ and find the largest value of e such that all jobs $J_1, \dots, J_k$ can be scheduled in $[e, D_j]$ under the condition that no job can start in a forbidden interval. Now, if $r_i \leq e < r_i + 1$, then $]e - 1, r_i[$ is declared as a forbidden region, since in any feasible schedule a job starting in $]e - 1, r_i[$ is not from $\{J_1, \dots, J_k\}$ and hence, the set $\{J_1, \dots, J_k\}$ is finished after D_j . After forming the set of all forbidden regions, the implementation of the earliest deadline scheduling rule gives an optimal schedule.

In [35], it has been shown that problem $P|p_j = 1, D_j, prec| -$ is unary NP-hard by a polynomial reduction from the 3-satisfiability problem. In [27], the authors have shown that even problem $P \mid p_j = 1, D_j = 3, prec \mid -$ is unary NP-hard. Note that problem $P \mid p_j = 1, D_j = 2, prec \mid -$ can be polynomially solved in $O(n)$ time. The important question about the complexity status of problem $P3 \mid p_j = 1, prec \mid C_{max}$ is still open whereas in [16], problem $P2 \mid p_j = 1, prec \mid C_{max}$ was solved in $O(n^3)$ time by a reduction of the problem to the maximum matching problem. In [7], an $O(n \log n)$

algorithm has been proposed for problem $P \mid r_j, p_j = p, \text{outtree} \mid C_{max}$ and it has been shown that problem $P \mid r_j, p_j = p, \text{intree} \mid C_{max}$ is unary NP-hard.

In [30], a polynomial algorithm with complexity $O(n^3 \log \log n)$ has been developed for problem $P \mid r_j, p_j = p, D_j \mid -$. The algorithm is based on an analysis of the structural properties of an optimal schedule. The main features of the optimal structure used in that paper are the following:

1. Any schedule is completely defined by the set of time slots $(1, t_1), (2, t_2), \dots, (n, t_n)$, where $i = 1, \dots, n$ is the slot number and t_i is the starting time of slot i .
2. If we know the set of all occupied time slots, then an optimal schedule can be constructed by the earliest deadline scheduling procedure.
3. If the earliest deadline scheduling procedure generates a sequence $(1, t_1), \dots, (k, t_k)$, such that job J_l processed in (k, t_k) is late, then job J_l cannot be scheduled in (k, t_k) and has to be scheduled earlier. To provide this, one of the slots $(1, t_1), \dots, (k-1, t_{k-1})$ has to be pulled right. To this end, Simons chooses the closest to (k, t_k) slot which is occupied by a job whose deadline exceeds D_l .

It has been shown that the use of these principles leads to the construction of an optimal schedule for problem $P \mid r_j, p_j = p, D_j \mid -$. The same algorithm solves the problems $P \mid r_j, p_j = p, D_j \mid C_{max}$ and $P \mid r_j, p_j = p, D_j \mid \sum C_j$.

In [32], an $O(mn^2)$ time algorithm was proposed for problems $P \mid r_j, p_j = p, D_j \mid -$, $P \mid r_j, p_j = p, D_j \mid C_{max}$, and $P \mid r_j, p_j = p, D_j \mid \sum C_j$. The algorithm is substantially based on the ideas from [17] and [30]. In [10], the following linear programming formulation was proposed for problem $P \mid r_j, p_j = p, D_j \mid -$:

$$\sum_{i=1}^z x_{ji} = p, \quad j = 1, \dots, n \quad (1)$$

$$\sum_{j=1}^n x_{j,i+1} + \dots + \sum_{j=1}^n x_{j,i+y} \leq mp, \quad i = 0, \dots, z-y \quad (2)$$

$$x_{ji} = 0 \text{ if } I_i \not\subseteq [r_j, D_j], \quad i = 1, \dots, z, \quad j = 1, \dots, n \quad (3)$$

$$0 \leq x_{ji} \leq p, \quad i = 1, \dots, z, \quad j = 1, \dots, n \quad (4)$$

Here x_{ji} is equal to the amount of job J_j processed in the interval I_i , where $\{I_i \mid i \in \{1, \dots, z\}\} = \{[r_j + kp, r_j + kp + p] \mid k \in \{\dots, -1, 0, 1, \dots\}\}$, and $y = \max\{k \mid I_{i+1} \cap \dots \cap I_{i+k} \neq \emptyset, i \in \{0, \dots, z-k\}\}$. In the above system, the polyhedron (1) and for each $i = 0, \dots, z-y$, with $i+ey \leq z$, the polyhedron (2) are integer. Nevertheless, their intersection is not an integer polyhedron and therefore, the obtained solution is not necessarily integer. Using an obtained solution, one can construct an optimal schedule in two equivalent ways, namely:

1. It is possible to find the intervals which are occupied in a feasible schedule. Then the earliest deadline scheduling procedure generates an optimal schedule.
2. It is possible to transform the obtained solution into the form $x_{ji}^* \in \{0, p\}$. The obtained vector gives an optimal solution for problem $P \mid r_j, p_j = p, D_j \mid -$.

In both cases, the following property of an optimal solution holds: If $k = \min\{i \mid x_{ji}^* \neq 0, j = 1, \dots, n\}$ for the optimal solution, then the time slot I_k is occupied in an optimal schedule.

It has been shown in [10] that, to solve problem $P|r_j, p_j = p, D_j| \sum C_j$, it is sufficient to solve the following linear programming problem:

$$\begin{aligned} & \text{Minimize } \sum_{i=1}^z \sum_{j=1}^n D(I_i) x_{ji} \\ & \text{subject to (1), (2), (3), (4).} \end{aligned}$$

Here $D(I_i)$ is the right endpoint of the interval I_i .

In [15], the proposed linear programming formulation was transformed by means of the substitution $y_t = \sum_{s=1}^t \sum_{j=1}^n x_{js}$ into the following form:

$$\begin{aligned} & \text{Minimize } \sum_{i=1}^z D(I_i)(y_i - y_{i-1}) \\ & \text{subject to } y_z - y_0 = n \\ & y_i - y_{i-1} \geq 0, \quad i = 1, \dots, n \\ & y_i - y_{i-y+1} \leq m, \quad i = y, \dots, n \\ & y_0 = 0. \end{aligned}$$

The authors have shown that this model can be solved in $O(n^4)$ time.

To solve problem $P|r_j, p_j = p| \sum w_j C_j$, it is sufficient to solve the following linear program (see [10]):

$$\begin{aligned} & \text{Minimize } \sum_{i=1}^z \sum_{j=1}^n w_j D(I_i) x_{ji} \\ & \text{subject to (1), (2), (3), (4).} \end{aligned}$$

In [11], it has been shown that, in order to solve problem $P|r_j, p_j = p| \sum T_j$, it is sufficient to

$$\begin{aligned} & \text{Minimize } \sum_{i=1}^z \sum_{j=1}^n \max\{0, D(I_i) - d_j\} x_{ji} \\ & \text{subject to (1), (2), (3), (4).} \end{aligned}$$

In [2], a polynomial time algorithm with the complexity $O(n^{6m+1})$ was proposed for problem $Pm | r_j, p_j = p | \sum w_j U_j$. The algorithm is based on the following observations and definitions:

1. It is possible to restrict the set of starting times by $\{r_j + zp \mid z \in \{0, \dots, n\}, j \in \{0, \dots, n\}\}$.
2. If the set of time slots $(1, t_1), \dots, (n, t_n)$ is known in advance for an optimal schedule, where $i = 1, \dots, n$ is a slot number and t_i is the starting time of slot i , then the desired schedule can be constructed by the earliest due date rule.
3. The only situation when job J_j follows J_i with $d_j < d_i$ is the situation when J_i is processed within $]r_j - p, r_j + p[$, i.e., the starting time of J_i is before r_j .
4. A profile is defined as a vector $(a_1, \dots, a_m)$, where $a_i \in \{r_j + zp \mid z \in \{0, \dots, n\}, j \in \{0, \dots, n\}\}$, and $\max\{a_1, \dots, a_m\} - \min\{a_1, \dots, a_m\} \leq p$. In an optimal schedule, for any job J_j scheduled at t_j , only a profile a with $t_j \in \{a_1, \dots, a_m\}$ can be prescribed.

5. $k[a, b]$ is defined as the set of early jobs from $\{J_1, \dots, J_k\}$ scheduled between the profiles a and b , and $W_k[a, b]$ is the maximal weight for such a set.

Now, dynamic programming can be applied by using the formulas

$$W_k[a, b] = \max\{W'_k[a, b], W_{k-1}[a, b]\} \text{ if } \max\{a_1, \dots, a_m\} - p \leq r_k < \min\{b_1, \dots, b_m\},$$

$$\text{and } W_k[a, b] = W_{k-1}[a, b] \text{ if } r_k \notin [\max\{a_1, \dots, a_m\} - p, \min\{b_1, \dots, b_m\}],$$

$$\text{where } W'_k[a, b] = \max_{\substack{\min\{x_1, \dots, x_m\} \in \\ [r_k, d_k - p]}} \{W_{k-1}[a, x] + w_k + W_{k-1}[x', b]\}.$$

In [12], an $O(n^5)$ algorithm was proposed for problem $1 \mid r_j, p_j = p \mid \sum U_j$. This algorithm is analogous to the algorithm from [2], however, the authors do not use $W_k[a, b]$ but $w_k[a, B]$, i.e., for the given w, k and a , the authors minimize B . For the single machine case, B is the length of the considered subschedule. Thus, they define:

1. $k[a, \cdot]$ is the set of jobs from $\{J_1, \dots, J_k\}$ such that $r_j \geq a$ holds,
2. $w_k[a, B]$ equals the minimal value B such that it is possible to execute w jobs from $k[a, \cdot]$ in the time interval $[a + p, b]$.

In [29], a polynomial time algorithm was proposed for problem $P \mid r_j, p_j = p, D_j \mid L_{max}$. It is based on a binary search technique and on the fact that the number of possible starting points is polynomial in the problem size.

In [13], the case with identical jobs and uniform parallel machines has been considered, i.e., when each machine has some given speed. It has been shown how to solve problems $Q \mid p_j = p \mid \sum \varphi_j$ and $Q \mid p_j = p \mid \max \varphi_j$ in $O(n^2)$ time, where $\max \varphi_j = \max_{1 \leq j \leq n} \varphi_j(C_j)$, and $\varphi_j, j = 1, \dots, n$, are non-decreasing functions in the job completion times. The authors also indicated how to reduce the complexity for classical criteria and solved problems $Q \mid r_j, p_j = p \mid C_{max}$ and $Q \mid r_j, p_j = p \mid \sum C_j$ in $O(n \log n)$ and $O(mn^{2m+1})$ time, respectively.

Note that currently the most interesting open problems are $P \mid r_j, p_j = p \mid \sum U_j$ and $P \mid r_j, p_j = p, D_j \mid \sum w_j C_j$.

2.2 Some generalizations

In [6], a dynamic programming approach was used to solve polynomially problem $Pm \mid r_j, p_j = p \mid \sum f_j$, where $\sum f_j$ is an objective function depending on the completion times C_j , such that f_j is non-decreasing and $f_i - f_k$ is monotonic. Note that both classical criteria $\sum w_j C_j$ and $\sum T_j$ can be described in such a way.

In [20], a linear programming approach was proposed for problem $P \mid r_j, p_j = p \mid \sum f_j$, where $\sum f_j$ is the objective function from [6], i.e., f_j depends on the completion time C_j , such that f_j is non-decreasing, and $f_i - f_k$ is monotonic, and for problem $P \mid r_j, p_j = p, D_j \mid \max \varphi_j$, where φ_j is any non-decreasing function in the completion time C_j . The classical scheduling criteria described as $\max \varphi_j(C_j)$ are the minimization of maximum lateness $L_{max} = \max_j \{C_j - d_j\}$ and maximum tardiness $\max_j \{T_j\} = \max_j \{L_j, 0\}$.

The approach for problem $P \mid r_j, p_j = p \mid \sum f_j$ is analogous to that from [11] and consists in minimizing

$$\sum_{i=1}^z \sum_{j=1}^n f_j(D(I_i)) x_{ji}$$

subject to (1), (2),

$$x_{ji} = 0 \text{ if } R(I_i) < r_j, \quad i = 1, \dots, z, \quad j = 1, \dots, n$$

$$x_{ji} \geq 0, \quad i = 1, \dots, z, \quad j = 1, \dots, n.$$

Here $R(I_i)$ is the left endpoint of the interval I_i .

A polynomial algorithm for problem $P \mid r_j, p_j = p, D_j \mid \max \varphi_j(C_j)$ can be briefly described as follows. Note that the number of intervals available for processing can be polynomially bounded. Therefore, the possible number of different values $\varphi_j(D(I_i))$ is polynomially bounded, too. Take any $F = \varphi_j(D(I_i))$ for some i and j . Consider the following feasibility problem:

$$\sum_{i=1}^z x_{ji} = p, \quad j = 1, \dots, n$$

$$\sum_{j=1}^n x_{j,i+1} + \dots + \sum_{j=1}^n x_{j,i+y} \leq mp, \quad i = 0, \dots, z - y$$

$$x_{ji} = 0 \text{ if } R(I_i) < r_j \quad i = 1, \dots, z, \quad j = 1, \dots, n$$

$$x_{ji} = 0 \text{ if } \varphi_j(D(I_i)) > F$$

$$x_{ji} \geq 0, \quad i = 1, \dots, z, \quad j = 1, \dots, n$$

It is possible to find a solution for the above problem such that $x_{j,i} \in \{0, p\}$. At the same time, the obtained solution can be considered as a solution for problem $P \mid r_j, p_j = p, D_j \mid \max \varphi_j(C_j) \leq F$. Applying the same procedure for all different values of $F = \varphi_j(D(I_i))$, we can choose the minimal value of F . Since the number of different values $\varphi_j(D(I_i))$ is polynomially bounded, the proposed algorithm is polynomial.

In [31], the following problem has been considered. With each of the n jobs, there is associated a set of intervals. Each interval has a starting time and a finishing time. All data are assumed to be integers. The goal is to construct a feasible schedule so that each job is processed only in one of the prescribed intervals. It has been shown that the considered problem is unary NP -hard for the case of one machine and two prescribed intervals for each job by a polynomial reduction from the 3-satisfiability problem.

In [19], the following problem has been considered. As before, for each job J_j , $j = 1, \dots, n$, a processing time $p_j = p$, a release date r_j , and a deadline D_j are given. Besides we suppose that the time interval $[\min_j \{r_j\}, \max_j \{D_j\}]$ is divided into several intervals $[t_1, t_2[, [t_2, t_3[, \dots, [t_{T-1}, t_T[,$ where $\min_j \{r_j\} = t_1 \leq t_2 \leq \dots \leq t_T = \max_j \{D_j\}$, such that for each interval $[t_g, t_{g+1}[$ the number of available machines m_{g+1} is known in advance. Note that we do not fix the concrete set of m_{g+1} machines, i.e., at two different points of $[t_g, t_{g+1}[$, one can use different sets of m_{g+1} machines. Preemption of processing is not allowed, i.e., the processing of any job started at time t on one of the identical machines will be completed at time $t + p$ on the same machine. We want to find a feasible schedule such that the maximal number of machines used by $J_1, \dots, J_n$ is minimal. The problem has been reduced to the following linear programming problem.

Minimize M

$$\text{subject to } \sum_{i=1}^z x_{ji} = p, \quad j \in \{1, \dots, n\}$$

$$\sum_{j=1}^n x_{j,i+1} + \dots + \sum_{j=1}^n x_{j,i+q} \leq \min\{M, m_{k+1}\}p,$$

$$\begin{aligned} \text{where } i &\in \{0, \dots, z-q\}, q \in \{1, \dots, y\}, \\ k &\in \{1, \dots, T-1\}, \\ I_{i+1} \cap \dots \cap I_{i+q} \cap [t_k, t_{k+1}] &\neq \emptyset \end{aligned}$$

$$x_{ji} = 0 \text{ if } R(I_i) < r_j, \quad i = 1, \dots, z, \quad j = 1, \dots, n$$

$$x_{ji} = 0 \text{ if } D_j < D(I_i), \quad i = 1, \dots, z, \quad j = 1, \dots, n$$

$$x_{ji} \geq 0, \quad i = 1, \dots, z, \quad j = 1, \dots, n.$$

If we set x_{ji} equal to the amount of job J_j processed in the interval I_i and M is the number of machines we want to minimize, then any feasible schedule for the scheduling problem under consideration can be described as a feasible solution of the above linear programming problem.

On the other hand, the solution (x^*, M^*) obtained for the linear programming problem can be transformed into an optimal solution of the scheduling problem considered in polynomial time.

3 Problems with preemptions

Now we consider the following basic problem. There are m parallel machines and n jobs, each requiring an identical execution time p . With each job J_j , there is associated a release time r_j . The processing of any job may be interrupted arbitrarily often and resumed later on any machine. The objective is to construct a feasible schedule so as to minimize a given criterion.

It has been shown in [34] that problem $P \mid p_j = 1, prec, pmtn \mid \sum C_j$ is unary NP-hard. In [24], the technique based on linear programming has been developed. The proposed approach allows one to solve problem $R \mid r_j, pmtn \mid L_{max}$ in polynomial time. Using a pseudopolynomial reduction from numerical matching with target sums, it has been proved that problem $P \mid p_j = p, pmtn \mid \sum w_j U_j$ is unary NP-hard [9]. It has been noted in [2] that the algorithm developed in [24] for problem $R \mid r_j, pmtn \mid L_{max}$ can be used to solve both problem $P \mid p_j = p, pmtn \mid \sum U_j$ and problem $Q \mid p_j = p, pmtn \mid \sum U_j$ in polynomial time. Problem $P \mid r_j, p_j = p, pmtn \mid \sum U_j$ is unary NP-hard [18]. It is interesting to note that problem $P \mid pmtn \mid \sum U_j$ is binary NP-hard [23], however, its complexity status under an unary encoding is still an open question.

Problem $P \mid r_j, p_j = p, pmtn \mid \sum C_j$ has been solved in [1]. It is possible to prove that for problem $P \mid r_j, p_j = p, pmtn \mid \sum C_j$, an optimal schedule can be found in the class of schedules, where each job J_j is processed on machine m only within an (possibly empty) interval $[S_{j,m}, C_{j,m}[$, such that $C_{j,m} \leq S_{j+1,m}$ for each m and $j < n$, and $C_{j,m} \leq S_{j,m-1}$ for each $m > 1$ and j .

Using such a structure of an optimal schedule, one can formulate the following linear program which will solve problem $P \mid r_j, p_j = p, pmtn \mid \sum C_j$ in polynomial time.

$$\begin{aligned} &\text{Minimize} \quad \sum_{j=1}^n C_{j,1} \\ &\text{subject to} \quad S_{j,m} \geq r_j, \quad j = 1, \dots, n \end{aligned}$$

$$\sum_{q=1}^m (C_{j,m} - S_{j,m}) = p, \quad j = 1, \dots, n$$

$$S_{j,m} - C_{j,m} \leq 0, \quad j = 1, \dots, n, \quad m = 1, \dots, m$$

$$C_{j,q} - S_{j,q-1} \leq 0, \quad j = 1, \dots, n, \quad q = 2, \dots, m$$

$$C_{j,q} - S_{j+1,q} \leq 0, \quad j = 1, \dots, n-1, \quad q = 1, \dots, m$$

Another idea was used in [21] for problem $Q \mid r_j, p_j = p, pmtn \mid \sum C_j$. Note that for problem $Q \mid r_j, p_j = p, pmtn \mid \sum C_j$, an optimal schedule can be found in the class of schedules, for which $C_1 \leq C_2 \leq \dots \leq C_n$ holds, i.e., there exists an optimal solution in which the completion times are in the same order as the release times.

Let s^* be an optimal schedule for problem $Q \mid r_j, p_j = p, pmtn \mid \sum C_j$ with $C_1(s^*) \leq \dots \leq C_n(s^*)$. Further we set $r_{n+1} = r_n + n \cdot \max_q \{p/s_q\}$, i.e., r_{n+1} is a time point after which no job will be processed. Each $C_j(s^*)$ belongs to some interval $[r_i, r_{i+1}]$. However, if we know for each $C_j(s^*)$ the corresponding interval $[r_i, r_{i+1}]$ such that $C_j(s^*) \in [r_i, r_{i+1}]$, then an optimal schedule can be easily found using a reduction to a network flow problem, see [25]. Thus, the main question is to know the interval $[r_i, r_{i+1}]$ for each $C_j(s^*)$ such that $C_j(s^*) \in [r_i, r_{i+1}]$. However, this difficulty can be avoided due to criterion $\sum C_j$. For any job J_j , let the time interval $[r_i, r_{i+1}]$ be such that $C_j \in [r_i, r_{i+1}]$. Taking into account that $r_1 = 0$, we obtain $C_j = (r_2 - r_1) + (r_3 - r_2) + \dots + (r_i - r_{i-1}) + (C_j - r_i)$. Due to this decomposition, we introduce the completion time of job J_j for each interval $[r_i, r_{i+1}]$.

For each job J_j with $j = 1, \dots, n$ and for each interval $[r_i, r_{i+1}]$ with $i = 1, \dots, n$, we define the value $C(J_j, r_i)$ such that $C(J_j, r_i) = C_j(s^*)$ if $C_j(s^*) \in]r_i, r_{i+1}[$, but if $C_j(s^*) \leq r_i$, then we set $C(J_j, r_i) = r_i$, and if $C_j(s^*) \geq r_{i+1}$, then we set $C(J_j, r_i) = r_{i+1}$. So, for each $i = 1, \dots, n$, the values $r_i \leq C(J_1, r_i) \leq \dots \leq C(J_i, r_i) \leq C(J_{i+1}, r_i) = \dots = C(J_n, r_i) = r_{i+1}$ define a partition of the interval $[r_i, r_{i+1}]$. In turn, each interval $[C(J_{j-1}, r_i), C(J_j, r_i)]$ is completely defined by the jobs processed in it. Thus, we denote by $v(J_k, M_q, J_j, r_i)$ the part (amount) of job J_k processed in the interval $[C(J_{j-1}, r_i), C(J_j, r_i)]$ by machine M_q , i.e., the total processing time of job J_k by machine M_q in the interval $[C(J_{j-1}, r_i), C(J_j, r_i)]$ equals $\frac{v(J_k, M_q, J_j, r_i)}{s_q}$ and for any job J_k , equality $\sum_{q=1}^m \sum_{i=1}^n \sum_{j=1}^n v(J_k, M_q, J_j, r_i) = p$ holds. The values $C(J_j, r_i)$, where $j = 1, \dots, n$, $i = 1, \dots, n$, and the values $v(J_k, M_q, J_j, r_i)$, where $i, j = 1, \dots, n$, $k = j, \dots, i$, $q = 1, \dots, m$, define a feasible solution of the following linear program. For convenience, we introduce $C(J_0, r_i) = r_i$.

$$\text{Minimize } \sum_{i=1}^n \left((C(J_1, r_i) - r_i) + \dots + (C(J_n, r_i) - r_i) \right)$$

subject to

$$r_i = C(J_0, r_i) \leq C(J_1, r_i) \leq \dots \leq C(J_{i+1}, r_i) = \dots = C(J_n, r_i) = r_{i+1},$$

$$\sum_{q=1}^m \frac{v(J_k, M_q, J_j, r_i)}{s_q} \leq C(J_j, r_i) - C(J_{j-1}, r_i),$$

$$\sum_{k=1}^n \frac{v(J_k, M_q, J_j, r_i)}{s_q} \leq C(J_j, r_i) - C(J_{j-1}, r_i),$$

$$\sum_{i=1}^n \sum_{j=1}^n \sum_{q=1}^m v(J_k, M_q, J_j, r_i) = p,$$

where $i = 1, \dots, n$, $j = 1, \dots, n$, $k = 1, \dots, n$, $q = 1, \dots, m$,

$$C(J_j, r_i) \geq 0, \quad i = 1, \dots, n, \quad j = 1, \dots, n,$$

and for $q = 1, \dots, m$

$$v(J_k, M_q, J_j, r_i) = 0, \quad i = 1, \dots, n-1, \quad j = i+1, \dots, n, \quad k = 1, \dots, n$$

$$v(J_k, M_q, J_j, r_i) = 0, \quad i = 1, \dots, n, \quad j = 1, \dots, i, \quad k = 1, \dots, j-1,$$

$$v(J_k, M_q, J_j, r_i) = 0, \quad i = 1, \dots, n-1, \quad j = 1, \dots, i, \quad k = i+1, \dots, n,$$

$$v(J_k, M_q, J_j, r_i) \geq 0 \quad i = 1, \dots, n, \quad j = 1, \dots, n, \quad k = 1, \dots, n.$$

The above formulation includes $O(mn^3)$ variables and constraints, i.e., this problem can be polynomially solved. In [21], it has been shown that an optimal solution of the above linear program can be used to obtain an optimal schedule.

Problem $P \mid r_j, pmtn \mid \sum C_j$ has been proved to be unary NP-hard by a reduction from 3-partition in [1]. Problem $P \mid r_j, p_j = p, pmtn \mid \sum w_j C_j$ is unary NP-hard [28]. Thus, the minimal open problems are $1 \mid r_j, p_j = p, pmtn \mid \sum w_j C_j$ and $P \mid r_j, p_j = p, pmtn \mid \sum T_j$.

4 Summary

In Table 1, we give an overview on the main results for classical criteria.

Acknowledgements This work was partially supported by the Alexander von Humboldt Foundation.

References

1. P. Baptiste, P. Brucker, M. Chrobak, C. Dürr, S.A. Kravchenko, and F. Sourd, The complexity of mean flow time scheduling problems with release times, *J. Scheduling*, 10, 139–146, 2007.
2. P. Baptiste, P. Brucker, S. Knust, and V. G. Timkovsky, Ten notes on equal-processing-time scheduling, *4OR*, 2, 111–127, 2004.
3. P. Baptiste, and P. Brucker, Scheduling equal processing time jobs: a survey, In Y.T. Leung, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 78–96, CRC Press LLC, Boca Raton, FR, 2004.
4. P. Baptiste, M. Chrobak, C. Dürr, W. Jawor, and N. Vakhania, Preemptive scheduling of equal-length jobs to maximize weighted throughput, *Oper. Res. Letters*, 32, 258–264, 2004.
5. P. Baptiste, Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times, *Journal of Scheduling*, 2 (6), 245–252, 1999.
6. P. Baptiste, Scheduling equal-length jobs on identical parallel machines, *Discrete Appl. Math.*, 103, 21–32, 2000.
7. P. Brucker, M. R. Garey, and D. S. Johnson, Scheduling equal-length tasks under tree-like precedence constraints to minimize maximum lateness, *Math. Oper. Res.*, 2, 275–284, 1977.

8. P. Brucker, S. Knust, Complexity results for scheduling problems, <http://www.mathematik.uni-osnabrueck.de/research/OR/class/>
9. P. Brucker, S.A. Kravchenko, Scheduling equal processing time jobs to minimize the weighted number of late jobs, *Journal of Mathematical Modelling and Algorithms*, 5(2), 143–165, 2006.
10. P. Brucker, S.A. Kravchenko, Scheduling jobs with equal processing times and time windows on identical parallel machines, *J. Scheduling*, 11, 229–237, 2008.
11. P. Brucker, S.A. Kravchenko, Scheduling jobs with release times on parallel machines to minimize total tardiness, Universität Osnabrück, Preprint Heft 258, 13 pp., 2005.
12. M. Chrobak, C. Dürr, W. Jawor, L. Kowalik, and M. Kurowski, A note on scheduling equal-length jobs to maximize throughput, *J. Scheduling*, 9, 71–73, 2006.
13. M.I. Dessouky, B.J. Lageweg, J.K. Lenstra, and S.L. van de Velde, Scheduling identical jobs on uniform parallel machines, *Statistica Neerlandica*, 44, 115–123, 1990.
14. C. Dürr, The scheduling zoo, <http://www.lix.polytechnique.fr/~durr/query/>
15. C. Dürr, M. Hurrand, Finding total unimodularity in optimization problems solved by linear programs, *Proc. of the 14th Annual European Symposium on Algorithms (ESA)*, 315–326, 2006.
16. M. Fujii, T. Kasami, and K. Ninomiya, Optimal sequencing of two equivalent processors, *SIAM J. Appl. Math.*, 17, 234–248, 1969.
17. M.R. Garey, D.S. Johnson, B.B. Simons, and R.E. Tarjan, Scheduling unit-time tasks with arbitrary release times and deadlines, *J. Comput.*, 10, 256–269, 1981.
18. S.A. Kravchenko, On the complexity of minimizing the number of late jobs in unit time open shop, *Discrete Appl. Math.*, 100, 127–132, 2000.
19. S.A. Kravchenko, F. Werner, Minimizing the number of machines in a unit-time scheduling problem, Otto-von-Guericke-Universität Magdeburg, FMA, Preprint 25/07, 8 pp., 2007 (to appear in *European J. Oper. Res.*, doi: 10.1016/j.ejor.2008.10.008).
20. S.A. Kravchenko, F. Werner, On a parallel machine scheduling problem with equal processing times, *Discrete Appl. Math.*, 157, 848–852, 2009.
21. S.A. Kravchenko, F. Werner, Preemptive scheduling on uniform machines to minimize mean flow time, *Comput. Oper. Res.*, Vol. 36(10), 2816–2821, 2009.
22. E.L. Lawler, Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the ‘tower of set’ property, *Math. Comput. Modelling*, 20(2), 91–106, 1994.
23. E.L. Lawler, Recent results in the theory of machine scheduling, In A. Bachem, *Mathematical Programming: the state of the Art*, 202–234, Springer, Berlin, 1982.
24. E. L. Lawler, J. Labetoulle, On preemptive scheduling of unrelated parallel processors by linear programming, *J. Assoc. Comput. Mach.*, 25(4), 612–619, 1978.
25. J. Labetoulle, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan, Preemptive scheduling of uniform machines subject to release dates, in H.R. Pulleyblank, *Progress in Combinatorial Optimization*, 245–261, Academic Press, New York, 1984.
26. J.K. Lenstra, A.G.H. Rinnooy Kan, and P. Brucker, Complexity of machine scheduling problems, *Ann. Discrete Math.*, 1, 343–362, 1977.
27. J.K. Lenstra, A.H.G. Rinnooy Kan, Complexity of scheduling under precedence constraints, *Oper. Res.*, 26, 22–35, 1978.
28. J.Y.-T. Leung, G.H. Young, Preemptive scheduling to minimize mean weighted flow time, *Information Processing Letters*, 34, 47–50, 1990.
29. B. Simons, A fast algorithm for single processor scheduling, *Proc. IEEE 19th Annual Symposium on Foundations of Computer Science (FOCS’78)*, 246–252, 1978.
30. B. Simons, Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines, *SIAM J. Comput.*, 12, 7–9, 1983.
31. B.B. Simons, M. Sipser, On scheduling unit-length jobs with multiple release time/deadline intervals, *Oper. Res.*, 32, 80–88, 1984.
32. B.B. Simons, M.K. Warmuth, A fast algorithm for multiprocessor scheduling of unit-length jobs, *SIAM J. Comput.*, 18, 690–710, 1989.
33. Z. Tian, C.T. Ng, T.C.E. Cheng, An $O(n^2)$ algorithm for scheduling equal-length preemptive jobs on a single machines to minimize total tardiness, *J. Scheduling*, 9, 343–364, 2006.
34. J.D. Ullman, Complexity of sequencing problems, in J.L. Bruno, E.G. Coffman, Jr., R.L. Graham, W.H. Kohler, R. Sethi, K. Steiglitz, and J.D. Ullman, *Computer and Job/Shop Scheduling Theory*, John Wiley & Sons Inc., New York, 1976.
35. J.D. Ullman, NP-complete scheduling problems, *J. Comput. System Sci.*, 10, 384–393, 1975.

Table 1: Polynomially solvable and NP-hard problems

Non-preemptive problems	
$1 r_j, D_j -$	unary NP-hard [26]
$P p_j = 1, D_j, prec -$	unary NP-hard [35]
$P p_j = 1, D_j = 3, prec -$	unary NP-hard [27]
$P p_j = 1, D_j = 2, prec -$	solvable in $O(n)$ [27]
$P r_j, p_j = p, D_j -$	solvable in $O(mn^2)$ [32]
$1 r_j, p_j = p, D_j, prec C_{max}$	solvable in $O(n \log n)$ [17]
$P2 p_j = 1, prec C_{max}$	solvable in $O(n^3)$ [16]
$P r_j, p_j = p, outtree C_{max}$	solvable in $O(n \log n)$ [7]
$P r_j, p_j = p, intree C_{max}$	unary NP-hard [7]
$P r_j, p_j = p, D_j C_{max}$	solvable in $O(mn^2)$ [32]
$Q r_j, p_j = p C_{max}$	solvable in $O(n \log n)$ [13]
$Q p_j = p C_{max}$	solvable in $O(n + m \log m)$ [13]
$Q p_j = p L_{max}$	solvable in $O(n \log n)$ [13]
$P r_j, p_j = p, D_j L_{max}$	solvable in $O(mn^4)$ [29]
$P r_j, p_j = p, D_j \sum C_j$	solvable in $O(mn^2)$ [32]
$Q r_j, p_j = p \sum C_j$	solvable in $O(mn^{2m+1})$ [13]
$Q p_j = p \sum C_j$	solvable in $O(n + m \log m)$ [13]
$P r_j, p_j = p \sum w_j C_j$	reducible to LP [10]
$Q p_j = p \sum w_j C_j$	solvable in $O(n \log n)$ [13]
$1 r_j, p_j = p \sum T_j$	solvable in $O(n^7)$ [6]
$P r_j, p_j = p \sum T_j$	reducible to LP [11]
$Q p_j = p \sum T_j$	solvable in $O(n \log n)$ [13]
$Q p_j = p \max w_j T_j$	solvable in $O(n \log^2 n)$ [13]
$1 r_j, p_j = p \sum U_j$	solvable in $O(n^5)$ [12]
$1 r_j, p_j = p \sum w_j U_j$	solvable in $O(n^7)$ [5]
$Pm r_j, p_j = p \sum w_j U_j$	solvable in $O(n^{6m+1})$ [2]
$Q p_j = p \sum w_j U_j$	solvable in $O(n \log n)$ [13]
Preemptive problems	
$P p_j = 1, prec, pmtn \sum C_j$	unary NP-hard [34]
$P r_j, pmtn \sum C_j$	unary NP-hard [1]
$Q r_j, p_j = p, pmtn \sum C_j$	reducible to LP [21]
$P r_j, p_j = p, pmtn \sum w_j C_j$	unary NP-hard [28]
$R r_j, pmtn L_{max}$	reducible to LP [24]
$1 r_j, p_j = p, pmtn \sum U_j$	solvable in $O(n \log n)$ [22]
$P r_j, p_j = p, pmtn \sum U_j$	unary NP-hard [18]
$P pmtn \sum U_j$	binary NP-hard [23]
$Q p_j = p, pmtn \sum U_j$	solvable in $O(n \log^2 n + mn \log n)$ [2]
$1 r_j, p_j = p, pmtn \sum w_j U_j$	solvable in $O(n^4)$ [4]
$P p_j = p, pmtn \sum w_j U_j$	unary NP-hard [9]
$1 r_j, p_j = p, pmtn \sum T_j$	solvable in $O(n^2)$ [33]

Recent advances in solving two-job shop scheduling problems

Extensions and improvements of the geometric approach

Riad Aggoune · Yazid Mati

Abstract This paper presents recent results on the solution of two-job shop scheduling problems with practical constraints. In particular, extensions and improvements of the geometric approach are proposed. The approach is first explained on the classical job-shop problem and direct extensions are then discussed. The second part of the paper describes a modification of the geometric approach that allows integrating practical constraints such as limited availability of machines, with the same computational complexity.

1 Introduction

The majority of scheduling problems encountered in production systems are known to be NP-hard. There however exist some special cases that can be solved polynomially. One of these special cases is the two-job shop problem denoted by $J | n = 2 | \Phi$ using the three fields notation adopted in [8]. In the two-job shop scheduling problem we try to produce a schedule of two jobs J_1 and J_2 on a set of m machines $M = \{M_1, M_2, \dots, M_m\}$, which minimizes a given criterion Φ . Each machine M_j is available during the whole planning horizon and can process at most one operation at a time. Each job must be performed according to a linear sequence of operations $O_{i,1}, O_{i,2}, \dots, O_{i,n_i}$ fixed by the technological process, where n_i is the number of operations of job J_i ($i = 1, 2$).

The two-job shop scheduling problem can be solved using a polynomial algorithm named the geometric approach. The first research on the geometric approach was done as early as 1955 [5], however the use of this approach for developing scheduling methods was done in the recent ten years. Therefore, the aim of this paper is twofold. First, we review the research results on the geometric approach. In particular complexity

Riad Aggoune
Public Research Centre Henri Tudor, 66, rue de Luxembourg, BP 144 L-4002 Esch-sur-Alzette,
G.D. Luxembourg
E-mail: Riad.Aggoune@tudor.lu

Yazid Mati
Al-Qassim University, College of Business and Economics, Almelaida, P.O. Box 6033, Kingdom
of Saudi Arabia.
E-mail: matie@qu.edu.sa

results on two-job shop problems with practical considerations encountered in production systems are presented. In the second part, we show that it is possible to modify the geometric approach such that with the same worst-case complexity, additional constraints can be considered.

The remaining of the paper is organized as follows. In Section 2 the geometric approach is explained on the classical job-shop problem. Section 3 presents extensions of the approach when practical constraints are integrated in the job-shop problem. In Section 4 we present the improvement of the classical approach. Section 5 concludes the paper.

2 Presentation of the geometric approach

In this section, we describe the geometric approach for minimizing the makespan in a two-job shop problem, give the complexity of this approach and finally present some improvements to make the approach more efficient. For sake of clarity, we use the following example to illustrate the developments of this section. There are two jobs to be scheduled on four machines: $J_1 = \{M_1(2), M_2(4), M_3(2), M_4(1)\}$ and $J_2 = \{M_3(1), M_1(2), M_2(3), M_4(2)\}$.

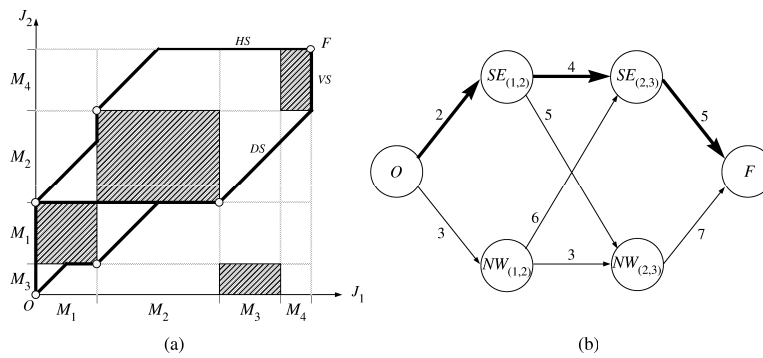


Fig. 1 The shortest path in the plane and the corresponding digraph

The idea behind the geometric approach consists in reducing the two-job shop scheduling problem in a shortest path one. The approach starts by representing the scheduling problem in a two-dimension plane with rectangular objects as obstacles defined as follows. Each job is associated to an axis (for example, x-axis to J_1 and y-axis to J_2). Each axis is decomposed in n_i sub-intervals according to the sequence of operations of the associated job. A sub-interval $I_{i,j}$ corresponds to operation $O_{i,j}$ and its length is equal to the processing time of that operation. A rectangle defined by sub-intervals I_{1,k_1} and I_{2,k_2} is an obstacle, denoted as (k_1, k_2) , if and only if operations O_{1,k_1} and O_{2,k_2} use the same machine. The upper and right boundaries of the rectangle defined by the origin O (which indicates the start of processing of the two jobs) and the final point $F = (\sum_{k=1}^{n_1} p_{1,k}, \sum_{k=1}^{n_2} p_{2,k})$ (which points out the finish of processing of the two jobs) is considered as a degenerated (or final) obstacle. In Figure 1 (a), the

rectangle defined by $I_{1,1}$ and $I_{2,2}$ is an obstacle as the first operation of J_1 and the second operation of J_2 are both performed on machine M_1 .

A feasible solution of the two-job shop scheduling problem corresponds to a path going from the origin O to the final point F , which avoids the interior of any obstacle (k_1, k_2) seeing that operations O_{1,k_1} and O_{2,k_2} are performed on the same machine and the preemption is not allowed. The path is composed of horizontal segments where only job J_1 is processed (segment HS in Figure 1 (a)), vertical segments where only job J_2 is processed (segment VS in Figure 1 (a)) and diagonal segments where jobs J_1 and J_2 are performed concurrently using different machines (segment DS in Figure 1 (a)). The length L of a path, which is equal to the duration of the associated schedule, is $L = \sum(\text{horizontal segments}) + \sum(\text{vertical segments}) + (1/\sqrt{2}) \sum(\text{diagonal segments})$.

Accordingly, the determination of the optimal schedule for the $J \mid n = 2 \mid C_{max}$ becomes a shortest path problem in the two-dimension plane, which consists of finding a shortest trajectory that connects the origin O to the final point F . This shortest path problem can equivalently be obtained in a digraph $G = (V, E, d)$ defined as follows. The set of nodes V is composed, in addition to the origin O and the final point F , of South-East (SE) and North-West (NW) corners of obstacles met when progressing diagonally in the plane. Each node $v_i \in V - \{F\}$ has at most two outgoing arcs. These arcs are obtained by progressing diagonally (45°) until an obstacle (k_1, k_2) is hit. If the obstacle (k_1, k_2) is the final obstacle, the node F is the only successor of v_i , otherwise the SE and NW corners of (k_1, k_2) are immediate successors of v_i . The nodes of the digraph are characterized by the index of operations that generate the obstacle. Indeed, the south-east corner of an obstacle (k_1, k_2) is denoted $SE_{(k_1, k_2)}$ and the north-west corner is denoted $NW_{(k_1, k_2)}$. The distance between v_i and a south-east (resp. north-west) corner is the projection along the x (resp. y) axis. The digraph of the illustrative example is given in Figure 1 (b)).

The complexity of the geometric approach for solving the two-job shop problem has been first calculated by [19] and [7]. Assuming that the function $\Phi(C_1, C_2)$ can be calculated in constant time, this complexity depends mainly on the construction of the digraph and the determination of a shortest path. The digraph G can be constructed in $O(r \log r)$ steps where r is the number of obstacles. Notice that r may be as large as $O(n_1 n_2)$ and if each job is only processed once on a machine then $r = \min(n_1, n_2)$. Moreover, seeing that the digraph G is acyclic, the determination of a shortest path can be obtained in $O(|V| + |E|)$ where $|V| = 2r$ and $|E| < 4r$. Consequently, the whole complexity is $O(r^2 \log r)$.

3 Integrating practical considerations

Since the works proposed in [19, 7], a number of extensions of the geometric approach has been proposed to deal with complex scheduling problems. In this section, we give the modifications of the geometric approach when practical constraints are introduced in the job-shop problem. All improvements of the geometric approach arise in the way the digraph is constructed.

3.1 Multi-resources operations

In some applications an operation may need a combination of more than one resource to be performed. For instance, an operation needs a machine and a human operator to load the job on the machine and to control if the operation is correctly started. The construction of the digraph in the geometric approach is exactly the same as described before and the only difference concerns the definition of the obstacles. A rectangle defined by sub-intervals I_{1,k_1} and I_{2,k_2} is an obstacle if and only if operations O_{1,k_1} and O_{2,k_2} share at least a same resource.

3.2 Minimizing regular criteria

The geometric approach can be extended to solve the two-job shop scheduling problem with any regular criterion $\Phi(C_1, C_2)$ where $C_i (i = 1, 2)$ is the completion time of job J_i . This extension has been proposed in [20]. The construction of the digraph is similar to the one of the makespan, and the only difference concerns the determination of the value of the criterion. In the case of the makespan, the optimal value corresponds to the length of the shortest path of F . In the case of a regular criterion, it is necessary to evaluate the criterion each time the upper or right boundary of the final obstacle is reached. Indeed, these boundaries corresponds to the end of a job (the processing of the two jobs is achieved in the final point F), and the optimal value depends on all points where paths from O to F hit the final obstacle. If a path hits the upper boundary at a point p , the value of the criterion of the schedule represented by this path is $\Phi(h(p) + d(p, F), h(p))$ where $h(p)$ is the length of the shortest path from O to p and $d(p, F)$ is the length of the horizontal segment between p and F . Similarly, if the right boundary is hit at a point q , the value of the criterion is given by $\Phi(h(q), h(q) + d(q, F))$ where $d(q, F)$ is the length of the vertical segment between q and F . The complexity of the new geometric approach is the same as for the makespan. The complexity of the approach does not change.

3.3 Optimizing non regular criteria

The analysis carried out in [20] for regular criteria has been extended in [4] for a class of non-regular criteria, which is the quasi-convex criteria of the completion times of the two jobs. The basic idea of the extended geometric approach consists in finding a set of non dominated schedules according to Pareto optimality, and then searching for a global optimum among these schedules. The complexity depends on the processing times of operations and is equal to $O(r \log r + \log H)$ time, where r is the number of obstacles and H is the maximum operation processing time.

The authors in [11] pursue the investigation of non-regular criteria and study the case when there is an earliness cost on the first operation and a tardiness cost on the last operation of each job. They first adapted the results of [20] for minimizing the tardiness when the starting times of both jobs are fixed. Then they use the algorithm of [20] as a subroutine for solving the problem when earliness and tardiness costs are imposing on each operation. The overall complexity of the algorithm is $O(r^2)$ where r is the number of obstacles.

3.4 Preemption of operations

The extension of the geometric approach for solving the job-shop problem when the preemption of operations is allowed has been proposed in [20]. In this extension, a feasible path from O to F can go inside an obstacle and the digraph is generalized through the addition of new nodes and the modification of the definition of the successors of a node. More precisely, if the lower boundary of an obstacle is hit, the south-east corner of the obstacle and the point A obtained vertically until reaching the upper boundary, are the two successors of the starting node v . Otherwise, if the left boundary of an obstacle is reached, the north-west corner and the point B obtained horizontally until reaching the right boundary, are the two successors of v . The complexity of the new approach is $O(r^3)$, where $r = \max\{n_1, n_2\}$.

3.5 Precedence between operations

Precedence relations $O_{1,k_1} \prec O_{2,k_2}$ (O_{1,k_1} before O_{2,k_2}) between two operations that are processed on the same machine, can be modeled in the two-dimension plane. This modeling has been proposed in [9] for developing lower bounds used in a branch and bound method for the job-shop problem. Notice that the precedence relations between operations correspond to the disjunctions already fixed in the disjunctive graph model. Since O_{1,k_1} must be started before O_{2,k_2} , every path from O to F that crosses the obstacle (k_1, k_2) at the left becomes forbidden. Thus, the obstacle (k_1, k_2) is expanded leftward beyond the y -axis until $X = 0^-$. Similarly, for precedence relations $O_{2,k_2} \prec O_{1,k_1}$, the obstacle has to be expanded beyond the x -axis until $Y = 0^-$. The complexity of the geometric approach remains unchanged.

3.6 Release dates of jobs and operations

The geometric approach can easily integrate the release date r_i of the J_i that means that the realization of J_i cannot start before r_i . This is performed by adding a fictitious operation before the first operation of the manufacturing process of J_i . The fictitious operation is performed on a fictitious machine not used by the operations of the two jobs, and the processing time is equal to r_i .

The release date $r_{i,j}$ of an operation $O_{i,j}$ is used to express the fact that $O_{i,j}$ cannot be started before $r_{i,j}$. To deal with these release dates in the geometric approach, the authors in [9] have used a third axis (z -axis) to express the time progress. It is worth mentioning that the scheduling problem studied in [9] is $J|n = 2; prec; r_i | \max(C_{i,j} + q_{i,j})$, where $C_{i,j}$ is the completion time of operation j of job J_i , and $q_{i,j}$ its tail. The scheduling problem is represented in a space and the obstacles becomes fences that must be avoided by any feasible path. The solution of the studied problem gives lower bounds for a branch and bound method for the job-shop problem. The complexity of the approach is at most $O(n^2(n + n_{\max}))$, where $n = O(n_1 n_2)$ and $n_{\max} = \max\{n_1, n_2\}$.

3.7 Blocking of machines

The blocking of machines in production systems ([15], [10]) is related to the lack of buffer space between machines. Indeed, the new production systems have more and

more tendency to build production lines with very limited storage capacity. Consequently, a job completing an operation cannot release the machine if the buffer is full and must wait until the machine needed for the next operation becomes available. The geometric approach can take into account the blocking of machines when solving the two-job shop problem. The extension is proposed in [?] who introduce new obstacles in the plane and generalize the construction of the digraph. The new obstacles are added to deal with the deadlock situations that represents states of the production system from which no job can be finished. The digraph is generalized by modifying the definition of the immediate successors of nodes. For any node v_i , all nodes that can be reached by first going diagonally and then vertically (resp. horizontally), are considered as successors of v_i . The complexity of the new extension is $O(n_1 n_2 (n_1 + n_2))$.

3.8 Machines availability

An extension of the geometric approach for the two-job shop scheduling problem with the unavailability of machines is proposed in [2]. Each machine is associated with a set of unavailability periods in which it cannot process operations. The starting and finishing times of the unavailability periods are known in advance. The authors assume that the operations are non-preemptive, which means that their executions can be interrupted neither by unavailability periods, nor by other operations. The new extension, called temporized geometric approach, is based on the definition and the introduction of new nodes in the digraph, as well as a new and dynamic way to progress from one node to its immediate successors. More precisely, in addition to the north-west and south-east corners of the obstacles, some nodes can be located between two lines bounding an operation. To characterize these nodes an additional information is added that consists in marking the duration already processed of the associated operation. A new progression method is also proposed to determine the immediate successors of a given node. The complexity of the new extension is $O(Ks^4)$ where $s = \max\{n_1, n_2\}$ and K is the maximum number, for all the machines, of unavailability periods.

4 Improved approach

We develop in this section an improved version of the classical geometric approach for problem $J, |n = 2| C_{max}$. It is based on the introduction of earliest dates for the vertices of the network associated with the problem. This characterization allows to integrate the evolution of time and thus to deal with practical constraints such as limited availability of the machines (see [2]). More precisely, we compute for each vertex v of the plane with obstacles the earliest date, noted $h(v)$, and which corresponds to the smallest duration to reach vertex v from the origin O (note that $h(O) = 0$). Thus, $h(v)$ is equal to the length of the shortest path going from O to v . Furthermore, we propose a new progression method that allows to integrating practical constraints on the execution of jobs without introducing additional points (i.e. without increasing the computational complexity of the approach). Indeed, we propose modified processing times in order to focus on the times needed to complete operations according to the considered constraints. It is thus necessary to modify the way the graphical representation is defined.

4.1 Proposed Representation

In the proposed approach we keep the classical representation in the plane with obstacles of the scheduling problem. However, the two axis are decomposed into unitary sub-intervals $I_{i,j}$ which length are no more proportional to the processing times of the associated operations $O_{i,j}$. Instead, we propose to divide the sub-intervals according to the processing times. As a consequence, the simultaneous execution of two operations $O_{1,j}$ and $O_{2,k}$ job J_1 and J_2 cannot be not represented by a diagonal leg (with an angle of 45 degrees), but follows a direction induced by the processing times of the involved operations. Let us consider the following example, illustrated in Figure 2, where one operation $O_{1,j}$ of job J_1 and two operations $O_{2,k}$ and $O_{2,k+1}$ of jobs J_2 are represented by unitary intervals. The processing times are respectively equal to $p_{1,j} = 4$, $p_{2,k} = 2$ and $p_{2,k+1} = 5$. Starting from vertex v , the simultaneous execution of $O_{1,j}$ and $O_{2,k}$ corresponds to a progression in the direction given by angle α .

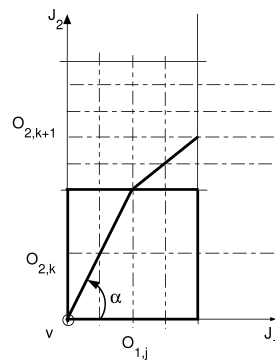


Fig. 2 New progression method

Since $p_{1,j} > p_{2,k}$, the execution of $O_{2,k}$ finishes before the one of $O_{1,j}$. Thus, the path crosses the upper boundary of the square corresponding to the operations. When operation $O_{2,k+1}$ can start, $\Delta_{1,j} = 2$ units of $O_{1,j}$ are already executed. As in the classical geometric approach, the minimal quantity between $p_{2,k+1}$ and $p_{1,j} - \Delta_{1,j} = 2$ can be executed before the path crosses the next horizontal or vertical line.

Angle α is such that the length of its opposite side $L_{2,k}$, which corresponds to the progression of job J_2 is:

$$L_{2,k} = \frac{\min(p_{1,j} - \Delta_{1,j}, p_{2,k} - \Delta_{2,k})}{p_{2,k}}, \quad (1)$$

where $\Delta_{2,k}$ and $\Delta_{1,j}$ represent respectively the portion of operation $O_{1,j}$ and $O_{2,k}$ already executed. Note that these values are both equal to 0 for vertices located at corners. However, it is necessary to evaluate the direction of the progression each time the path crosses a horizontal or vertical line. In fact in this case only one operation is to be started, the other one being partially executed.

The length $L_{1,j}$ of the adjacent side of angle α , which corresponds to the execution of job J_1 is given by:

$$L_{1,j} = \frac{\min(p_{1,j} - \Delta_{1,j}, p_{2,k} - \Delta_{2,k})}{p_{1,j}}. \quad (2)$$

Finally, angle α is defined by:

$$\tan \alpha = \frac{L_{2,k}}{L_{1,j}} = \frac{p_{1,j}}{p_{2,k}}. \quad (3)$$

In the example considered in Figure 2, the path progressing from v first goes in direction $\alpha = \arctan(\frac{p_{1,j}}{p_{2,k}}) = \arctan(\frac{4}{2}) \approx 63deg$. Then, the direction is given by $\arctan(\frac{p_{1,j}}{p_{2,k+1}}) = \arctan(\frac{4}{5}) \approx 39deg$.

Figure 3 shows the new representation for the problem given by the example considered in section 2. Numbers represent the earliest dates associated with the vertices. We can see that the set of vertices is the same as in Figure 1. Indeed, the new representation respects the rates of executions between operations of the two jobs. Pathes hit the obstacles at the same portions of operations. The main difference with the classical graphical approach is that it is no more possible to read the distances between vertices by (orthogonal) projections of the legs on the axis. However, we are more interested in the time needed to reach a vertex v' starting from vertex v , than in the geometrical distance between them. That information is already included in the definition of the earliest dates of the vertices. The general distance, which is the length of the shortest path between two vertices v and v' is thus defined by formula $\bar{d}(v, v') = h(v') - h(v)$.

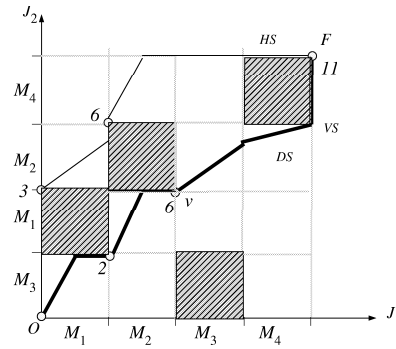


Fig. 3 General representation

4.2 Benefits of the new approach

The main advantage from using this new representation is that it allows dealing with time-dependant processing times and thus easily integrate practical constraints, such as limited availability of the machines, in the scheduling problem. In fact, the only additional step is to consider modified processing times for operations that are not fixed in advance but depend on their starting times (which are given by the earliest

dates $h(v)$ of the considered vertices). Given a vertex v with earliest date $h(v)$, we compute the time needed to complete next operations of job J_1 and J_2 .

When the modified processing times $\tilde{p}_{1,j}$ and $\tilde{p}_{2,k}$ are calculated for operations $O_{1,j}$ and $O_{2,k}$ of jobs J_1 and J_2 respectively, the progression is done in the direction given by angle α such that:

$$\tan \alpha = \frac{\tilde{p}_{1,j}}{\tilde{p}_{2,k}}. \quad (4)$$

In case of unavailability constraints, the modified processing times include the initial ones and possibly an iddle time (corresponding to the time preceding an availability period) and the duration of an availability period. For instance, let suppose that machine M_3 is unavailable between times $S_{3,1} = 7$ and $S_{3,1} + P_{3,1} = 9$ in the previous example. The first part of the shortest path given by Figure 3 remains the same. If we consider the last two operations of jobs and vertex v which earliest date is $h(v) = 6$, then processing time of operation $O_{1,3}$ of job J_1 becomes $\tilde{p}_{1,3} = (S_{3,1} - h(v)) + P_{3,1} + p_{1,3} = (7 - 6) + 2 + 2 = 5$. The other processing times remain unchanged. Thus, the path from vertex v goes first in the direction given by angle $\alpha = \arctan(\frac{\tilde{p}_{1,3}}{\tilde{p}_{2,3}}) = \arctan(\frac{5}{3})$. Then the path crosses the horizontal line corresponding to the last operation of job J_2 . The new direction for the path is given by $\alpha = \arctan(\frac{\tilde{p}_{1,3}}{\tilde{p}_{2,4}}) = \arctan(\frac{5}{2})$. The next Figure 4 shows the modifications of the shortest path.

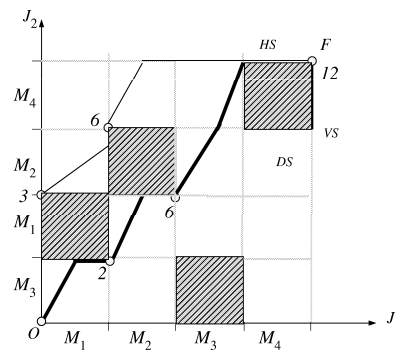


Fig. 4 Integration of availability constraints

Finally, that approach can be applied to other two-job shop scheduling problems where processing times, or more precisely, the times to complete operations are not fixed in advance.

5 Conclusion

We have proposed in this paper a comprehensive state-of-the art on the application of the geometric approach for solving shop scheduling problems with two jobs. First, the approach has been described on the job-shop problem for minimizing the makespan, followed by extensions that integrate some real-life considerations. A modification of the original approach for solving more complex problems were then presented.

References

1. R. Aggoune. *Shop scheduling with machines unavailability*. PhD Thesis, University of Metz, France, 2002.
2. R. Aggoune and M-C. Portmann. Flow shop scheduling problem with machine availability: a heuristic approach. *International Journal of Production Economics*, 99: 4–15, 2006.
3. A. Agnetis and G. Oriolo. The machine duplication problem in a job shop with two jobs. *International Transactions on Operational Research*, 2(1), 45–60, 1995.
4. A. Agnetis, P.B. Mirchandani, D. Pacciarelli and A. Pacifici. Job shop scheduling with two jobs and nonregular objective functions, *INFOR*, 39(3), 227–244, 2001.
5. S. Akers. A graphical approach to production scheduling problems. *Operations Research*, 4:244–245, 1956.
6. S. Akers and J. Friedman. A non-numerical approach to production scheduling problems. *Operations Research*, 3:429–442, 1955.
7. P. Brucker. An efficient algorithm for the job-shop problem with two jobs. *Computing*, 40:353–359, 1988.
8. P. Brucker. *Scheduling Algorithms*. Springer-Verlag, Berlin Heidelberg, 1998.
9. P. Brucker and B. Jurisch. A new lower bound for the job-shop scheduling problem. *European Journal of Operational Research*, 64:156–167, 1993.
10. N. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44:510–525, 1996.
11. Y. Hendel and F. Sourd. Job-shop with two jobs and irregular criteria. *International Journal of Operations Research*, 5:61–67, 2008.
12. W. Hardgrave and G. Nemhauser. A geometric model and a graphical algorithm for a sequencing problem. *Operations Research*, 11:889–900, 1963.
13. B. Jurisch. *Scheduling jobs in shops with multi-purpose machines*. PhD thesis, University of Osnabrück, Germany, 1992.
14. Y. Mati, N. Rezg, and X. Xie. A taboo search approach for deadlock-free scheduling of automated manufacturing systems. *Journal of Intelligent Manufacturing*, special issue on Metaheuristics, 12(5/6), 535–552, 2001.
15. Y. Mati. *Scheduling problems in automated manufacturing systems: models, complexity and solution methods*. PhD Thesis, University of Metz, France (in French).
16. Y. Mati and X. Xie. A polynomial algorithm for a two-job shop scheduling problem with routing flexibility. *IEEE International Conference on Robotics & Automation*, Taiwan, 2003.
17. Y. Mati and X. Xie. The complexity of the two-job shop scheduling problems with multi-purpose unrelated machines. *European Journal of Operational Research*, 152:159–169, 2004.
18. Y. Mati and X. Xie. A genetic search guided greedy algorithm for multi-resource shop scheduling with resource flexibility. *IIE Transactions*, to appear.
19. Y. Sotskov. Optimal servicing two jobs with a regular criterion. In: *Automation of Designing Processes*, Minsk, 86–95 (in Russian).
20. Y. Sotskov. The complexity of shop-scheduling problems with two or three jobs. *European Journal of Operational Research*, 53:326–336, 1991.
21. W. Szwarc. Solution of the Akers-Friedman scheduling problem. *Operational Research*, 8(6):782–788, 1960.

Minimizing the Makespan and Total Flow Time on the Permutation Flow Shop Scheduling Problem

E. Mokotoff

Abstract We present a method based on the Multi-Objective Simulated Annealing techniques. Results of the computational experiment indicate that the proposed algorithms provide efficient solutions with little computational effort. For all the tested instances a net set of potentially efficient schedules has been obtained and compared with previously published results.

1 Introduction

We present two algorithms based on a MOSA scheme (named as MOSAD and MOSAI) to minimize the makespan and total flow time on the classic permutation flow shop scheduling problem. This scheduling problem belongs to the set of hard Combinatorial Optimization problems (CO), even considering only one criterion. It is common to find real problems that could be modeled by an instance of this problem, *e.g.* on production lines.

In the permutation flow shop problem all jobs must pass through all machines in the same order, so that a feasible schedule is any job sequence. Moreover, quality is, in real-life, a multidimensional notion. A schedule is valued on the basis of a number of criteria, for example: makespan, flow-time, completion-time, tardiness, inventory, utilization, etc. If only one criterion is taken into account, no matter what criterion is considered, some aspect of the quality of the schedule will result regardless. Thus, we have to add the additional complexity that comes from the multivariant condition of corresponding alternative schedules. A solution which is optimal with respect to a given criterion might be a poor candidate for another. Of course, for multicriteria scheduling problems expecting to find the optimum schedule must usually be discarded. We would be satisfied with finding a set of Pareto optimal alternatives, and we have to let some subjective considerations intervene, such as the decision-maker preferences. It is actually a Multicriterion Decision Making problem (MDM), and at the present time, there is no other rational tool to apply to discard alternatives. Only with the breakthrough of metaheuristics in solving CO problems did researchers begin to adapt metaheuristics to solve Multi-Objective Combinatorial Optimization (MOCO) problems.

Multi-Objective Simulated Annealing (MOSA) methods are metaheuristics based on Simulated Annealing (SA) to tackle MOCO problems. SA, introduced by Kirkpatrick et al. [29], has demonstrated its ability in solving combinatorial problems such as vehicle routing, production scheduling, timetabling, etc. [30]. [49] presents a broad study of the application of

Ethel Mokotoff
Universidad de Alcalá
ethel.mokotoff@uah.es

SA to MOO. Based on a MOSA scheme, we have developed our algorithms to provide the decision-maker with efficient solutions for the scheduling problem we are dealing with.

The proposed methods consist in providing the decision-maker with approximate efficient solutions for this scheduling problem. Starting with a set of initial sequences, the algorithm samples a point in its neighbourhood. If this generated sequence is dominated, we still accept it with a certain probability. Different constructive and heuristic algorithms are used to compute initial good sequences and lower bounds for the different criteria. We carried out an intensive computational experiment by making use of the 90 benchmark problems given by Taillard [56]. The procedure is good enough to give efficient solutions with little computational effort. The performance analysis includes a set of metrics specific for evaluating Multi-Objective Optimization algorithms (MOO). The presented algorithms have been developed after an extensive study involving the influence of the number of initial solutions, the neighborhood search procedure and the SA parameters, together with the CPU time. With these experiments we have obtained a net set of potentially efficient schedules and we have updated some published net sets, for the same instances.

Thus, this paper is organized as follows: In the next section we will briefly introduce the MOCO theory and notations (section 2). In section 3, the classical permutation flow shop problem statement is presented, and its computational complexity and previous results are looked at. Section 4 describes the new approaches based on the MOSA scheme. Section 5 reports on the computational experiment. We conclude, in section 6, with a summary discussion on further research.

2 Multi-Objective Combinatorial Optimization problem

An appropriate schedule can not be obtained unless one observes the whole set of important criteria. The multidimensional nature of the problem at hand leads us to the area of MultiCriteria Optimization (see [13], for a state of the art review). In general for MOO problems the solutions vary according to the criterion considered individually. If the criteria are not conflicting, it is possible to obtain a global optimal solution. In the vast majority of cases, they do conflict and thus, the knowledge of the decision-maker preferences is necessary to solve the problem.

Due to the *NP-hardness* of the most CO problems just considering only one regular criterion (except for restricted special cases), research work concerning them have been intensively developed. Even though MDM, as well as CO, have been intensively studied by many researchers for many years, it is surprising that a combination of both, *i.e.* MOCO, was not widely studied until the last decade, as it is not long since interest in this field has been shown [12]. The proliferation of metaheuristic techniques has encouraged researchers to apply them to this highly complex context.

Approximation for MOO is a research area which has gained increasing interest in recent years. Multi-objective metaheuristics seek an approximate set of Pareto optimal solutions. The main question is how to ensure that the obtained non-dominated set covers the Pareto front as widely as possible. The first published methods were adaptations of single-objective optimization. Nowadays they have their own entity. They were initially inspired by evolutionary algorithms or neighborhood search. Furthermore, recent developments are more hybridized, giving rise to multi-objective hyperheuristic methods. A hyperheuristic can be thought of as a heuristic method, which iteratively selects the most suitable heuristic from amongst many [6].

The problem of obtaining a uniformly distributed set of non-dominated solutions is of great concern in Pareto optimization. The specification of the search direction, by tuning weights, is the method that directly attempts to drive the current solution towards the desired region of the trade-off frontier. Hyperheuristic approaches attempt to do it by applying the neighbourhood search heuristic that is more likely to drive the solution in the desired direction. This technique can be applied to single-solution and population-based algorithms.

Most of the published works in MOO are a priori methods since they assume that the decision-maker preferences can be expressed. The hierarchical approach penalizes too much the less important criteria, while setting a criterion as the most important one. In reality, the decision-maker preferences are usually smooth, giving less importance to the main criterion and more to the less important criteria. Considering a composed function of the criteria involved in the problem, it is implicitly assumed that the decision-maker preferences are accurately reflected in this objective function. The decision-maker knows the preferable schedule, but it is not easy to express this preference in a function. In general, a priori approaches give a solution to the problem, which cannot usually be trusted to be the most preferred solution.

To be confident with a particular solution to a problem with multiple objectives, the decision-maker's active involvement is required. In interactive methods, she indicates her preferences during the process of solution, guiding the search direction. [1] proposes an interactive *particle-swarm* metaheuristic for MOO. The approach presented by [26] can be placed between the a priori and interactive procedures because, despite including interaction, it is based on the assumption that decision-maker preferences are already relatively well-defined at the beginning of the solution process.

For methods that should offer the complete set of efficient solutions (a posteriori approaches), it is guaranteed that no potential preferable solution has been eliminated, but the number of efficient solutions can be overwhelmingly high to warrant proper examination by the decision-maker.

We refer the reader to:

- [12, 58] for further information on MOCO theory;
- [31, 28] for overviews on metaheuristic applications;
- [25] for metaheuristics for bi-criteria optimization problems.

For each particular metaheuristic:

- Multi-Objective Genetic Algorithms (MOGA), to [2] and [25]; general Evolutionary Multi-Objective Algorithms, to [9] and [17].
- MOSA, to [49], [57], [20], and [34].
- Multi-Objective Tabu Search, to [15].

3 Permutation Flow Shop Scheduling Problem

In a permutation flow shop scheduling problem, there are jobs and machines or stages. Each job needs to complete one operation on the machines during a fixed processing time and all jobs must pass through all machines in the same order. The scheduling process involves just finding the sequence of jobs that optimizes one or more criteria. But, the computational complexity usually grows exponentially with the number of machines making the problem intractable even if only one criterion is considered.

3.1 Notation

We will use the notation that follows:

J : set of n jobs J_i ($i=1, \dots, n$)

M : set of m machines M_j ($j=1, \dots, m$)

p_{ij} : processing time of job J_i on machine M_j

C_i : completion time of job J_i

C_{max} : the maximum completion time of all jobs J_i . This is the schedule length, which is also called the makespan.

F_i : flow time of job J_i , $F_i = C_i - r_i$

The optimal value of any criterion is denoted with an asterisk, e.g. C_{max}^* denotes the optimal makespan value.

We will use the three-parameter notation, $\alpha/\beta/\gamma$, introduced by [18] and extended for [52] to MultiCriteria scheduling problems. The first field specifies the machine environment (F represents general permutation flow shop); the second, job characteristics; and the third refers to the chosen optimality criterion for single criteria models, and it extends to cover multicriteria as well as methodology.

3.2 NP-Complexity and state of the art review

The classical permutation flow shop problem, like almost all deterministic scheduling problems, belongs to the wide class of CO problems, many of which are known to be NP-hard [16]. Since the early Johnson algorithm [27] that solves $F2//C_{max}$ in polynomial time, only a few restricted cases have been shown to be efficiently solvable. Minimizing the sum of completion times is still NP-complete for two machines [16]. Real-life scheduling problems require more than one criterion. Nevertheless, the complex nature of flow shop scheduling has prevented the development of models with multiple criteria. Permutation flow shop scheduling research has been mostly restricted to the treatment of one objective at a time. Furthermore, attention has focused on the makespan criterion. However, the total flow time performance measure has also received some attention. These two regular criteria constitute a conflicting pair of objectives [46]. Specifically, the total flow time criterion is a work in process inventory performance measure, and the makespan criterion is equivalent to the mean utilization performance measure. While total flow time is a customer-oriented performance measure, the makespan criterion is a firm-oriented performance measure. Therefore, the set of efficient solutions to a bi-criteria model that seeks to optimize both measures simultaneously would contain valuable trade-off information crucial to the decision-maker, who has to identify the most preferable solution, according to her preferences.

Solving a bi-criteria model for a general number of machines implies heavy computational requirements, since both criteria makespan and total flow time lead to NP-hard problems even when they are treated individually. Due to the fact that only the $F2//C_{max}$ problem can be solved in polynomial time (the rest of flow shop scheduling problems are *NP-complete*), research production concentrates on heuristics and enumerative approaches. The majority of research on bi-criteria flow shop problems concerns the two-machine case, in which some combination of $\sum C_i$ and C_{max} has to be minimized. Since $F2//\sum C_i$ is NP-hard in the strong sense, any multi-criterion approach including $\sum C_i$ will be NP-hard too.

[43], [40], [19] and [53], present heuristic procedures for the two-machine flow shop problem, where total flow time has to be minimized among the schedules that minimize makespan (lexicographical approach). Local search algorithms based on Ant Colony Optimization have been proposed by [54]. [22] presents a technique named Local Dynamic Programming.

[53] presents a branch and bound algorithm, which can solve problem instances of up to 35 jobs to optimality. [38] and [50] present heuristics and branch and bound algorithms for the $F2//f(\sum C_i, C_{max})$ problem. For the two-machine flow shop scheduling problem of

minimizing makespan and sum of completion times simultaneously, [47] presents an *a posteriori* approach based on branch and bound. [10] presents a branch and bound algorithm for the $F2//f(C_{\max}, T_{\max})$, and a heuristic to approximate the set of non-dominated solutions for the more general $F//f(C_{\max}, T_{\max})$ problem. [33] presents branch and bound algorithms for the $F2//f(C_{\max}, \sum U_i)$ and $F2//f(C_{\max}, \sum T_i)$ problems.

[48] and [60] consider a linear combination of makespan and flow time. [11] presents a comparison of four iterative improvement techniques for flow shop scheduling problems that differ in local search methodology. These techniques are iterative deepening, random search, tabu search and genetic algorithms. The evaluation function is defined according to the gradual satisfaction of explicitly represented domain constraints and optimization functions. The problem is constrained by a greater variety of antagonistic criteria that are partly contradictory.

[21] and [45] propose heuristic procedures for the general m machine case, considering $\sum C_i, C_{\max}, \sum I_j$. They are based on the idea of minimizing the gaps between the completion times of jobs on adjacent machines (one of our proposed improvements was inspired by this paper). [61] applies Ant Colony Optimization to the same problem.

[4] presents a MOGA that improves the previous MOGA presented by [51]. [37] presents a MOGA considering the makespan, total flow time and total tardiness, based on a weighted sum of objective functions with variable weights. This algorithm belongs to the class of Evolutionary Multi-Objective Optimization algorithms and [24] shows that this algorithm can be improved by adding a local search procedure to the offspring. [7] applies subpopulation GA to the same problems. Artificial chromosomes are created and introduced into the evolution process to improve the efficiency and the quality of the solution. [3] proposes a MOGA algorithm with preservation of dispersion in the population, elitism, and use of a parallel bi-objective local search so as to intensify the search in distinct regions. The algorithm is applied to the makespan-maximum tardiness and makespan-total tardiness problems. [14] investigates both *a priori* and *a posteriori* heuristics. The last one uncovers non-dominated solutions by varying the weight criteria in an effective way. [8] proposed a GA algorithm for the $F//f(\sum C_i, C_{\max})$ problem based on the concept of *gradual priority weighting* (the search process starts in the direction of the first selected objective function, and progresses such that the weight for the first objective function decreases gradually, and the weight for the second objective function increases gradually). [59] applies a similar idea. [42] presents a Pareto GA with Local Search, based on ranks that are computed by means of crowding distances. [42] and [59] apply the same initial population and improvement schemes.

[17] presents a study of the problem structure and the effectiveness of Local Search Neighborhoods within an evolutionary search framework on Multi-Objective flow shop scheduling problems.

[5] applies Dantzig-Wolfe reformulation and Lagrangian relaxation to an Integer Programming formulation to minimize a total cost of job function that includes: earliness, tardiness and work in process inventory costs.

4 MOSAD and MOSAI Algorithms

Based on MOSA methods, we have developed two variants of an approximation algorithm to get the Pareto solution set of the bi-criteria permutation flow shop problem, considering the minimization of the makespan and the total flow time simultaneously. MOSA methods are characterized by:

- An acceptance rule for new solutions, with some probability that depends on the temperature level.
- A scheme of cooling.
- A mechanism for browsing the efficient frontier.
- Information is obtained from the set of solutions.

4.1 The main procedure

The proposed algorithms mainly consist in a MOSA scheme which begins with an initial iterate solution, X_0 , that belongs to a set S of initial points (feasible solutions of $F/(C_{\max}, \sum C_i)$), which are good solutions for one of the two simplified single-criterion versions of the problem). X_0 is then sampled with a point Y in its neighbourhood. But instead of accepting Y , if it is better than the current iterate regarding an objective function, we now accept it, if it is not dominated by the current solution. In this case, we make Y the current iterate, add it to the Potentially Efficient solution set (PE), and throw out any point in PE that is dominated by Y . On the other hand, if Y is dominated by X_0 , we still make it the current iterate with some probability. This randomization is introduced in the procedure to attempt to reduce the probability of getting stuck in a poor locally optimal solution.

The sampling procedure is iterated according to the MOSA scheme described later. The solutions that are generated, during the optimization process, make iterative updates to the PE point set, to get closer to the Pareto optimal set (E). The only complicated aspect of this algorithm is the necessity of generating solutions in several directions of the bi-objective space search, $(C_{\max}, \sum C_i)$. So, to be able to cover the entire efficient frontier, a diversified set of points must be generated. Neighbourhood search procedures play a crucial role in the performance of the algorithm.

At each time during the search, the selection of the next heuristic to be used is based on the quality of the current seed. A set of simple neighbourhood exploration heuristics has been developed. Then, the approach proposed here selects the most appropriate neighbourhood heuristic at certain points during the search, in order to uncover the solutions in the Pareto optimal front.

The objective function values of the MOCO problem plays here the role of acceptance rule. The discrete nature of the problem at hand makes it possible that some efficient solutions do not minimize any aggregated function of the criteria. Only supported efficient solutions will be admitted for entrance into the PE. In order to avoid the non-supported efficient solutions to enable entrance into the PE, we have developed a bi-objective model where both criteria are included in a domination control. For just bi-criteria models, checking whether a solution is dominated by another is not computationally costly, and besides, updating the non-dominated solution set have to be tackled in any case (for more than two criteria models, the use of aggregated functions is absolutely justified).

To get a set of feasible initial solutions, S , the solutions obtained by means of several constructive and heuristic algorithms are considered. For each initial solution X_0 , the following procedure is applied.

- Initialization ($X_q = X_0, N_{count} = q = 0$)
- Improvement
- Iteration n
- Sample a neighbour Y
- Improvement
- Evaluate Y
- If Y is acceptable: $X_q = Y$ ($N_{count} = 0$). Else, we accept the solution with probability
- $$p = \exp\left(-\frac{\Delta\phi}{T_q}\right)$$
- $$X_{q+1} = \begin{cases} \leftarrow \frac{p}{1-p} Y, & N_{count} = 0 \\ \leftarrow X_q, & N_{count} = N_{count} + 1 \end{cases}$$
- Update PE.
- $q = q + 1$. If $q \pmod{N_{step}} = 0$, then $T_q = \alpha T_{q-1}$, else $T_q = T_{q-1}$. If $N_{count} = N_{stop}$ or $T_q < T_{stop}$, then stop. Else iterate.

Improvement of the initial solutions and neighbouring generation are carried out by simple neighbourhood exploration heuristics. The objective of these procedures is to approximate the trade-off surface in a more efficient way by using those movements that are more promising according to the quality of the current solution. These procedures induce a privileged direction of search to the efficient frontier. So, to be able to cover the entire efficient frontier, a diversified combination of initial solutions and neighbouring generation heuristics must be considered.

In the MOSA scheme described previously, a neighbouring solution of the current permutation must be chosen. The most important neighbourhoods based on a single permutation as an input are:

- Exchange, swapping the positions of two jobs at i and k , with $i \neq k$. The remaining jobs in the sequence conserve their positions.
- Insertion, forward or backward shift, removing a job at i and reinserting it at a different position k , with $k > i$ in forward case, and with $k < i$ in backward case. The remaining jobs in the sequence must be re-arranged in order to keep their relative positions.

In the following each of the steps of this generic scheme, specifications and parameters are described in detail.

Initial solutions

The first two solutions are obtained by means of the two simple but effective constructive algorithms: The improvement on NEH [39] by Taillard [55], looking for the minimum C_{max} , and the algorithm for the $F // \sum C_i$ problem presented by Rajendran [44]. This common list scheduling procedure is also applicable re-combining: *making list* (ordering by p_i or w_i), and *criterion to be minimized* (makespan or total flow time). A third solution is obtained following the NEH algorithm, only changing the minimization criterion, i.e. in order to minimize the partial total flow time. By analogy, Rajendran algorithm is applied to obtain another solution, a good solution for the makespan criterion (these four initial solutions are used by the MOGA algorithm presented in [42]). All of these four generation algorithms share the same four-step structure.

In the computation of the initial solutions, the procedure keeps the useful data in order to save computational effort (e.g. job lists, best partial schedules, etc.). With this technique it is possible to obtain a selective list of efficient solutions as seeds, instead of just randomly-generated ones.

When sampling solutions, only those which pass the domination control are taken into account for listing in the PE solutions set. The rest of the generated solutions are only used as input (for improvement or neighbouring search) and discarded later.

Improvement Scheme

Instead of inducing the search direction by tuning weights, to improve the distribution of non-dominated solutions we apply different neighbouring search heuristics.

While insertion has been shown to lead to superior results compared with exchange, for flow shop scheduling problems with C_{max} objective [55], it does not seem to be possible to derive a similar general rule when considering total flow time criterion. So, for improving makespan, we have only implemented insertion. On the other hand, for flow time, we try with insertion and exchange.

In the valuation of a neighbour, it is very important to save computational effort in order to check a larger neighbourhood. When inserting or exchanging jobs in a schedule, it should be possible to discard some dominated candidate permutations with just small computational

requirements examining the corresponding partial schedules. With this in mind, we have developed two neighbouring generation heuristics: one devoted to the search for neighbouring solution superior than the current one regarding makespan; another one sampling better solutions according to the total flow time measure. Depending on the generation procedure employed for the seed, the neighbouring heuristic is chosen. The first one is applied to the seeds X_1 and X_4 , while the second one is applied to the seeds X_2 and X_3 .

Improving Makespan: In order to reduce the search space, we have developed a technique based on elimination by domination conditions. Furthermore, we compute the lower bound for the makespan introduced by [56]. Thus, if we find out a permutation having this makespan value, we stop searching on the C_{max} axis, and concentrate effort in exploring the direction of reducing $\sum C_i$, in the neighborhood of C_{max}^* . The lower bound is computed as:

$$LB(C_{max}) = \max_j (\min_i \sum_{k=1}^{j-1} p_{ik} + \sum_{i=1}^n p_{ij} + \min_i \sum_{k=j+1}^m p_{ik}) \quad (1)$$

The algorithm for the $F // C_{max}$ problem (improvement over NEH), by Taillard [55], is actually a procedure to compute the value of the partial makespan when a job i is added in a partial schedule at position k . We employ this algorithm embedded in our neighbourhood search heuristic as a shortcut to evaluate a partial permutation. So, we do not need to compute the objective function for the complete schedule. Based on domination criteria for partial schedules (see [23] and [35]), we have developed our elimination neighbouring search.

Any partial schedule of t jobs, $Jp(t) = \{J_1, J_2, \dots, J_t\}$, where $t=1, 2, \dots, n$, is a sequence of the indices corresponding to the jobs in $Jp(t)$, and it could be named as $\sigma I_{Jp(t)}$. The completion time for a partial schedule $\sigma I_{Jp(t)}$ on machine k , where $k=1, 2, \dots, m$, is denoted by $C(\sigma I_{Jp(t)}, k)$. It was proved that:

Elimination Criterion 1 If $C(\sigma II_{Jp(t)}, k) \leq C(\sigma I_{Jp(t)}, k)$ for $k=1, 2, \dots, m$, then $\sigma II_{Jp(t)}$ dominates $\sigma I_{Jp(t)}$.

For the case where $\sigma II_{Jp(t)}$ and $\sigma I_{Jp(t-1)}$ are partial schedules of $Jp(t) \supset Jp(t-1)$, being $Jp(t-1) = \{J_j\}$, $\Delta k = C(\sigma II_{Jp(t)}, k) - C(\sigma I_{Jp(t-1)}, k)$ is defined. It was proved that:

Elimination Criterion 2 If $\Delta k - 1 \leq \Delta k \leq p_{jk}$ for $k=2, 3, \dots, m$, then $\sigma II_{Jp(t)}$ dominates $\sigma I_{Jp(t)}$ or $\sigma I_{Jp(t-1)}$.

Both theorems allow us to discard any completion of a partial schedule $\sigma I_{Jp(t)}$ or $\sigma I_{Jp(t-1)}$, because a schedule at least as good exists among the completion of another partial schedule $\sigma II_{Jp(t)}$.

The improvement scheme proposed in this section is based on the sequential insertion of a job in the current sequence at each possible different position. Since jobs with larger total processing time at the beginning of the schedule bring, in general, schedules with less makespan value, the proposed scheme selects, for insertion, a subset of jobs which are located at the first $\beta\%$ of the total positions in the current sequence. Hence, the set of t jobs scheduled at $\{1, 2, \dots, t\}$ positions, in the current permutation, σ_{Xq} , where $t = n \beta\%$, is selected for exploration consisting in checking whenever a better partial permutation, involving these t jobs, could be built.

Theoretically, we have to check and compare, for each job placed at i on σ_{Xq} , with $i=1$ to t , the makespan that results when this job is placed at a different position j , with $j=1$ to t . Nevertheless, the elimination criteria described above leads to efficiency gains. The Elimination Criterion 2 will filter any potential permutation generated by moving a job for which, to be placed at a different position with respect to its position in the current schedule, will not yield a sequence with less total completion time. Only for a potential permutation that passes this control, specified for a job to be moved, we check then for the different positions. By the Elimination Criterion 1, any potential schedule σ_y , for which the current schedule σ_{Xq} is at least as good, will be discarded. If one partial schedule is not eliminated, then the

corresponding complete schedule, σ_Y , becomes the generated neighbouring solution, $\sigma_{Xq} = \sigma_Y$, and the lower bounds used for computations are updated.

Improving Total Flow Time: Following the ideas of [21] and [45], we have developed an improvement heuristic looking for permutations with lower total flow time values, but attempting not to loss the level obtained in makespan. The original idea was to minimize gaps between successive operations that would lead to a better quality solution. The pair of jobs with the most positive gaps has to be placed at the beginning, while the pair of jobs with the most negative gaps has to be placed at the end of the schedule. During the total processing of the whole set of jobs, the larger gaps would have more chance of being compensated with the negative gaps corresponding to the pair of jobs scheduled at the end of the sequence.

The improvement scheme proposed in this section is based on the interchange of adjacent pairs of jobs with positive gaps in the current permutation. Since the objective is to minimize gaps, the jobs are listed in descending order of gaps. Exchanging adjacent jobs with larger gaps is more likely to result in a permutation which yields less flow time value. The procedure selects, for exchanging, a subset of jobs, $Jp(t)$, which are located at the later $\beta\%$ of the total positions in the current sequence σ_{Xq} . The exploration consists in checking whenever a new permutation obtained by exchanging an adjacent pair of jobs of $Jp(t)$, yields a schedule with lower flow time value.

Step 1: The subset of jobs placed at the last t positions of σ_{Xq} , where $t = \beta\%n$, is selected to constitute the set $Jp(t)$.

Step 2: For the jobs of $Jp(t)$, the gaps between every pair of adjacent jobs in σ_{Xq} , is then computed as

$$G_i = \sum_{j=1}^m p_{ij} - \sum_{j=1}^m p_{i+1,j}, \text{ for } i=n-t, n-t+1, \dots, n-1 \quad (2)$$

Step 3: Jobs in $Jp(t)$ are listed in descending order of gaps G_i . Jobs with negative gaps are not included in the list, and ties are broken in descending order of this similar gap:

$$G'_i = \sum_{j=1}^m (m-j+1)p_{ij} - \sum_{j=1}^m (m-j+1)p_{i+1,j}, \text{ for } i=n-t, n-t+1, \dots, n-1 \quad (3)$$

that is computed only in the case of a tie.

Step 4: The first job in the list, $J\sigma_{Xq}(i)$, scheduled at position i in the current permutation σ_{Xq} , will be set at $i+1$, in a new permutation σ_{II} , and its counterpart, the job placed at $i+1$ in σ_{Xq} , will be set at position i in σ_{II} .

Step 5: If $\exists j / C_{i,j}(\sigma_{II}) + C_{i+1,j}(\sigma_{II}) < C_{i,j}(\sigma_{Xq}) + C_{i+1,j}(\sigma_{Xq})$, or $C_{i,m}(\sigma_{II}) < C_{i+1,m}(\sigma_{Xq})$, then σ_{II} is accepted as a new permutation, $\sigma_{Xq} = \sigma_{II}$, then return to Step 1. Otherwise proceed to Step 6.

Step 6: Remove $J\sigma_{Xq}(i)$ from the list of jobs. If the list is not exhausted, then return to Step 4.

Updating Potential Efficient Set

When a neighbouring solution Y is accepted and made the current solution, $X_q = Y$, the set of PE solutions should be updated. If X_q is a new non-dominated solution, it should be added to the archive set and the set should be updated. Any solution dominated by the new one will be removed from the set. For the efficiency of this algorithm the updating PE process is crucial.

In order to save computational effort in updating PE, non-dominated solutions are always stored in ascending order of one of the criterion values, thus their other criterion values will be stored in descending order. This arrangement constitutes a fast method of finding out dominated instances with respect to a new solution, and of updating the PE.

4.2 Perturbation of X_k

Starting with the improved solutions, this procedure aims to help in approximating the trade-off surface in a more efficient way. Depending on the relation between the optimization criterion for which the iterate solution presents the least deviation (which coincides, in general, with the minimizing criterion for which it has been calculated) and the criterion taken into account for the improvement technique, we have differentiated two movement strategies which give rise to the two variants of the proposed method.

The common procedure explores the neighbourhood of the current solution, X_q , and yields a neighbour Y , following the direct search strategy (MOSAD) or the inverse search strategy (MOSAI) as follows: Let z_i be the preferable criterion according to the chosen strategy. For the permutation $\sigma_{X_q} = \{I_1, I_2, \dots, I_n\}$, where I_i , with $i=1, 2, \dots, n$, is the index of the job scheduled at the position i in σ_{X_q} , we will check its neighbourhood for finding out a better neighbour with respect to z_i . As it is known that insertion brings better improvement than exchange, this procedure generates potential permutations by inserting, forward and backward, removing each job I_i , and reinserting it in a different position at random. For each job, I_i , where $I_i \neq I_1$, and $I_i \neq I_n$, the procedure will choose randomly two positions for insertion. One position to its right, choosing randomly a number between $i+1$ and n , for forward insertion, and another position to its left, choosing randomly a number between 1 and $i-1$, for backward insertion. For jobs in extreme positions, I_1 and I_n , only one direction of insertion can be chosen. For I_1 only forward insertion is possible to apply, hence, to select a new position, a random number between 2 and n must be generated. In a similar way, I_n can only be inserted at positions to its left, so a number between 1 and $n-1$ has to be chosen. Thus, the z_i value of the $2(n-1)$ potential permutations has to be evaluated and the permutation with minimum z_i , is then selected as the neighbouring solution Y . This mechanism has been inspired on the rule known as Random Insertion Perturbation Scheme, introduced by [41]. The development of MOSAI was motivated for the observation (in previous studies) of a tendency of solutions to be concentrated near the extreme points of the efficient frontier. MOSAI intensifies the random generation of seeds, filling in the search space with more central solutions (see Figure 1).

5 Computational Experiments

To know the performance of the proposed algorithms which have been described we have tested on benchmark problems from [56]. The benchmark set contains 90 test instances of 9 different sizes, with the number of jobs equal to 20, 50 and 100, and the number of machines equal to 5, 10 and 20. For each size, a sample of 10 instances is presented. Thus, we have solved these 90 instances with MOSAD and MOSAI. The quality of the obtained approximations has been analyzed regarding the following measures:

$$Q_1(PE) = \frac{|PE \cap N|}{|PE|} 100\% \quad (4)$$

$$Q_2(PE) = \frac{|PE \cap R|}{|R|} 100\% \quad (5)$$

$$C(A, B) = \frac{|b \in B / \exists a \in A : a \prec b|}{|B|} \quad (6)$$

$$Dist1_R(PE) = \frac{1}{|R|} \sum_{x \in R} \left\{ \min_{y \in PE} \{d(X, Y)\} \right\} \quad (7)$$

$$Dist2_R(PE) = \max_{X \in R} \left\{ \min_{Y \in PE} \{d(X, Y)\} \right\} \quad (8)$$

where R is a reference set (the ideal R is the set of efficient solutions), N is a net set of non-dominated solutions, $N = \{X \text{ is Pareto efficient in } (PE \cup R)\}$, which should be at least as good as R (see [36]).

With the PE for the 90 instances obtained by means of the proposed algorithms, added to the algorithms presented in a previous work [36], we have built a net set to be used as R for this computational experiment and we have plot the efficient frontier for all of the instances.

We have also updated the net sets for the case published in both papers [59] and [42], size instance 50x20; and the net sets published in [59], size instance 20x20 and 100x20. Therefore, we have updated the net sets based on results from:

- Net of MOSA1, MOSA2, GPWGA [8], *a posteriori* [14], MOGLS [24], ENGA [4], published in [59], size instance 20x20, 50x20 and 100x20.
- Net of PGA-ALS, MOGLS, ENGA, GPWGA, published in [42], size instance 50x20.
- PE of the proposed algorithms, MOSAD and MOSAI, and PE of the previous algorithms [36], size instance 20x5, 20x10, 20x20, 50x5, 50x10, 50x20, 100x5, 100x10 and 100x20.

Table 1 shows the net set of non-dominated solutions corresponding to the ten problem instances of size 50x20 (the only size for which both papers [42] and [59], reports on net set). Tables 2 and 3 show the net set corresponding to the ten instances of sizes 20x20 and 100x20, respectively, combining our results with the net sets published in [59]. The superiority of MOSA2 in yielding the greatest number of non-dominated solutions is evident. However, MOSA2 becomes the worst algorithm when looking at computational effort information. A comparative study of the computational effort, in terms of the number of sequences generated during the entire search process, shows that MOSA2 is computationally the most demanding algorithm. Table 4 shows the number of sequences enumerated, for the proposed algorithms: for all the problem instances considered, and the data reported in [59] relative to sequences enumerated. In comparison with the rest of the algorithms MOSA2 demands twice as much effort as MOSA1, 3x that of MOGLS, and thirty times when comparing with the proposed algorithms, on average. Because of limited space only some comments on results obtained for the computed metrics are presented.

MOSAI yields more solutions which are kept in the final net set (heuristically efficient) than MOSAD, as much considering Q_1 (relative to [PE]) as Q_2 (relative to $[R]$).

With respect to the $C(\text{MOSAD}, \text{MOSAI})$ and $C(\text{MOSAI}, \text{MOSAD})$ measures (that make clear how many solutions provided by MOSAD are dominated by solutions from MOSAI, and vice versa), on average, we can affirm that MOSAI is better than MOSAD (only for the instance problems of 100x10 MOSAD shows a higher value of C than MOSAI). From the 90 problem instances, while MOSAI weakly outperforms MOSAD in 27 cases, MOSAD weakly outperforms MOSAI in only 2 cases. However the ouperformance of MOSAI over MOSAD does not apply to all the non-cardinal measures like $Dist1_R$, $Dist2_R$, and $Dist2_R/Dist1_R$. Though figures of $Dist1_R$, and $Dist2_R$ are still disadvantaged for MOSAD, the values of $Dist2_R/Dist1_R$ make it clear that MOSAD presents a more uniform distribution of solutions from PE over R . The corresponding efficient frontier show that, in spite of not yielding as many efficient solutions in R , the set of solutions found by MOSAD achieves two desirable aspects:

- It presents better (uniform in most cases) distribution in the bi-objective space.
- A wide range of values for each criterion is presented.

In spite of having a high percentage of non-efficient solutions, MOSAD gives a wide set of near-efficient solutions. Moreover, another advantage of MOSAD is the fact that it demands less effort than MOSAI, and it is remarkably less demanding than all the algorithms with solutions in the obtained net sets.

From the analysis above, it is possible to conclude that the influence of the improvement technique is crucial for the efficiency of the output. While MOSAD (with a direct search improvement) gives a larger number of solutions which help to improve the diversification of

the output with less computational effort, MOSAI (with an inverse search improvement) yields more efficient solutions.

6 Conclusions

In this paper we have presented two new algorithms based on MOSA techniques for a hard multicriteria scheduling problem. Starting with initial permutations obtained by single criteria constructive algorithms, improvements are made by computing lower bounds on the partial scheduling of neighbors, reducing the objective search space. The selection is made according to the criterion that is preferred at each iteration.

Due to the complexity of evaluating the performance of multi-objective algorithms, a set of different metrics have been computed, considering different attributes of each method. Furthermore, net sets of non-dominated solutions for the benchmarks problems of Taillard [56] have been obtained. After an extensive computational analysis, we can conclude that this kind of approach is appropriate to warrant a quick approximation output, which can serve as input for an interactive procedure. The search process should continue in the direction of the decision-maker's preferences. The proposed algorithms yield a set of potentially efficient solutions requiring less computational effort than other metaheuristic algorithms that have been published in the last few years.

It is not realistic to hope for general meta-optimization methods for every MOCO problem. Results of the analysis above give support to the hypothesis which states that specially-developed algorithms, combining general metaheuristic techniques, for specified combinatorial problems, perform better than general methods.

The development of methaheuristics and hyperheuristics for multi-objective methods is still a growing field. The presented algorithms have been proved to be useful, each having their own advantages and disadvantages. We are working now on the application of these multi-objective optimization approaches to environment problems.

Acknowledgements This research was in part supported by the Research Projects ECO2008-05895-C02-02, Ministerio de Ciencia e Innovación, Spain. The author is indebted to Paul Alexander Ayres for his help in the correct use of English.

References

1. Agrawal S, Dashora Y, Tiwari MK et al Interactive Particle Swarm: A Pareto-Adaptive Metaheuristic to Multiobjective Optimization. *IEEE T Syst Man Cy A* vol 38, 2:258-277 (2008)
2. Aickelin U Genetic Algorithms for Multiple-Choice Problems. PhD Thesis. University of Wales, Swansea (1999)
3. Arroyo J, Armentano V Genetic local search for multi-objective flowshop scheduling problems. *Eur J Oper Res* 167:717-738 (2005)
4. Bagchi TP Multiobjective Scheduling by Genetic Algorithms. Kluwer Academic Publishers, Dordrecht (1999)
5. Bülbül K, Kaminsky P, Yano C Flow shop scheduling with earliness, tardiness, and intermediate inventory holding costs. University of California, Berkeley (2003)
6. Burke EK, Landa-Silva JD, Soubeiga E Hyperheuristic Approaches for Multiobjective Optimization. In: *Proceedings of the 5th Metaheuristics International Conference*, 2003, Kyoto (2003)
7. Chang PC, Chen SH, Liu CH Sub-population genetic algorithm with mining gene structures for multiobjective flowshop scheduling problems. *Expert Syst Appl* 33:762-77 (2007)

8. Chang PC, Hsieh J-C, Lin SG The development of gradual priority weighting approach for the multi-objective flowshop scheduling problem. *Int J Prod Econ* 79:171–183 (2002)
9. Coello C, Mariano C Algorithms and Multiple Objective. In: Ehrgott M, Gandibleux X (eds) *Multiple Criteria Optimization. State of the Art Annotated Bibliographic Surveys*. Kluwer Academic Publishers, Boston (2002)
10. Daniels RL, Chambers RJ Multiobjective flow-shop scheduling. *Nav Res Log* 37:981-995 (1990)
11. Dorn J, Girsch M, Skele G et al Comparison of iterative improvement techniques for schedule optimization. *Eur J Oper Res* 94:349-361 (1996)
12. Ehrgott M, Gandibleux X Multiobjective Combinatorial Optimization: Theory, Methodology, and Applications. In: Ehrgott M, Gandibleux X (eds) *Multiple Criteria Optimization: State of the Art Annotated Bibliographic Surveys*. Kluwer Academic Publishers, Boston (2002)
13. Ehrgott M, Wiecek M Multiobjective Programming. In: Figueira J, Greco S, Ehrgott M (eds) *Multiple Criteria Decision Analysis*. Springer, New York (2005)
14. Framinan JM, Leisten R, Ruiz-Usano R Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation. *Eur J Oper Res* 141:559–569 (2002)
15. Gandibleux X, Mezdaoui N, Fréville A A tabu search procedure to solve multiobjective combinatorial optimization problems. *Lect Notes Econ Math* 455:291-300 (1997)
16. Garey MR, Johnson DS *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco (1979)
17. Geiger M On operators and search space topology in multi-objective flow shop scheduling. *Eur J Oper Res* 181:195–206 (2007)
18. Graham RL, Lawler EL, Lenstra JK et al Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann Discrete Math* 5:287-326 (1979)
19. Gupta JND, Neppalli VR, Werner F Minimizing total flow time in a two-machine flowshop problem with minimum makespan. *Int J Prod Econ* 69(3):323-338 (2001)
20. Hapke M, Jaskiewicz A, Slowinski R Interactive Analysis of multiple-criteria project scheduling problems. *Eur J Oper Res* 107(2):315-324 (1998)
21. Ho JC, Chang Y-L A new heuristic for the n -job, m -machine flowshop problem. *Eur J Oper Res* 52:194–202 (1991)
22. Huang G, Lim A Fragmental Optimization on the 2-Machine Bicriteria Flowshop Scheduling Problem. In: *Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence* (2003)
23. Ignall E, Schrage LE Application of the branch-and-bound technique to some flow-shop scheduling problems. *Oper Res* 13:400-412 (1965)
24. Ishibuchi H, Murata T A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE T Syst Man Cy C* 28(3):392-403 (1998)
25. Jaskiewicz A A Comparative Study of Multiple-Objective Metaheuristics on the Bi-Objective Set Covering Problem and the Pareto Memetic Algorithm. *Ann Oper Res* 131(1-4):135-158 (2004)
26. Jaskiewicz A, Ferhat AB Solving multiple criteria choice problems by interactive trichotomy segmentation. *Eur J Oper Res* 113(2):271-280 (1999)
27. Johnson SM Optimal two- and three-stage production schedules with setup times included. *Nav Res Log* 1:61-68 (1954)
28. Jones DF, Mirrazavi SK, Tamiz M Multi-objective meta-heuristics: An overview of the current state of the art. *Eur J Oper Res* 137:1-9 (2002)
29. Kirkpatrick S, Gelatt CD, Vecchi MP Optimization by simulated annealing. *Science* 220:671-680 (1983)
30. van Laarhoven PJM, Aarts EHL *Simulated Annealing: Theory and Practice*. Kluwer Academic Publishers, Dordrecht (1987)
31. Landa-Silva JD, Burke EK, Petrovic S An Introduction to Multiobjective Metaheuristics for Scheduling and Timetabling. *Lect Notes Econ Math*. 535: 91-129 (2004)

32. Lawler EL, Lenstra JK, Rinnooy Kan AHG Sequencing and scheduling: Algorithms and complexity. Handbooks in Operations Research and Management Science 4, Logistics of Production and Inventory, North-Holland, Amsterdam, 445-524 (1993)
33. Liao CJ, Yu WC, Joe CB Bicriterion scheduling in the two-machine flowshop. *J Oper Res Soc* 48:929-935 (1997)
34. Loukil T, Teghem J, Tuytens D Solving multi-objective production scheduling problems using metaheuristics. *Eur J Oper Res* 161:42–61 (2005)
35. McMahon GB Optimal Production Schedules for Flow Shop. *Can Oper Res Soc J* 7:141-151 (1969)
36. Mokotoff E. Multi-Objective Simulated Annealing for Permutation Flow Shop Problems. In: Chakraborty U, (ed) Computational Intelligence in Flow Shop and Job Shop Scheduling. Springer (in press).
37. Murata T, Ishibuchi H, Tanaka H Multi-Objective Genetic Algorithm and its Applications to Flowshop Scheduling. *Comp Ind Eng* 30(4):957–968 (1996)
38. Nagar, Haddock J, Heragu SS Multiple and bicriteria scheduling: A literature survey. *Eur J Oper Res* 81:88-104 (1995)
39. Nawaz M, Ensco EE Jr, Ham I A heuristic algorithm for the m machine, n job flowshop sequencing problem. *OMEGA-Int J Manage S* 11:91–95 (1983)
40. Neppalli VR, Chen CL, Gupta JND Genetic algorithms for the two-stage bicriteria flowshop problem. *Eur J Oper Res* 95:356-373 (1996)
41. Parthasarathy S, Rajendran C An experimental evaluation of heuristics for scheduling in a real-life flowshop with sequence-dependent setup times of jobs. *Int J Prod Econ* 49:255–263 (1997)
42. Pasupathy T, Rajendran C, Suresh RK A multi-objective genetic algorithm for scheduling in flow shops to minimize the makespan and total flow time of jobs. *Int J Adv Manuf Technol* 27:804–815 (2006)
43. Rajendran C Two-stage flowshop scheduling problem with bicriteria. *J Oper Res Soc* 43(9):879-884 (1992)
44. Rajendran C Heuristic algorithm for scheduling in a flowshop to minimize total flowtime. *Int J Prod Econ* 29:65–73 (1993)
45. Rajendran C Heuristics for scheduling in flowshop with multiple objectives. *Eur J Oper Res* 82:540-555 (1995)
46. Rinnooy Kan AHG Machine Scheduling problems: Classification, Complexity and Computations. Martinus Nijhoff, The Hague (1976)
47. Sayin S, Karabati S A bicriteria approach to the two-machine flow shop scheduling problem. *Eur J Oper Res* 113:435-449 (1999)
48. Selen WJ, Hott DD A mixed-integer goal-programming formulation of the standard flowshop scheduling problem. *J Oper Res Soc* 12/37:1121-1128 (1986)
49. Serafini P Simulated annealing for multiple objective optimization problems. In: Proceedings of the Tenth International Conference on Multiple Criteria Decision Making, Taipei (1992)
50. Sivrikaya-Serifoglu FS, Ulusoy G A bicriteria two machine permutation flowshop problem. *Eur J Oper Res* 107:414-430 (1998)
51. Srinivas N, Deb K Multiobjective function optimization using nondominated sorting genetic algorithms. *Evol Comp* 2(3):221-248 (1995)
52. T'kindt V, Billaut J-C Multicriteria scheduling: Theory, Models and Algorithms, 2nd edn. Springer, Berlin (2006)
53. T'kindt V, Gupta JND, Billaut J-C Two machine flowshop scheduling problem with a secondary criterion. *Comp Oper Res* 30(4):505-526 (2003)
54. T'kindt V, Monmarche N, Tercinet F et al An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. *Eur J Oper Res* 142(2):250-257 (2002)
55. Taillard E Some efficient heuristic methods for the flow shop sequencing problem. *Eur J Oper Res* 47:67–74 (1990)

56. Taillard E Benchmark for basic scheduling problems. *Eur J Oper Res* 64:278–285 (1993)
57. Ulungu EL Optimisation Combinatoire MultiCritère: Détermination de l'ensemble des solutions efficaces et méthodes interactives. PhD Thesis. Université de Mons-Hainaut, Mons (1993)
58. Ulungu EL, Teghem J Multiobjective Combinatorial Optimization problems: A survey. *J Multicriteria Dec Anal* 3:83–104 (1994)
59. Varadharajan TK, Rajendran C A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. *Eur J Oper Res* 167:772–795 (2005)
60. Wilson JM Alternative formulation of a flow shop scheduling problem. *J Oper Res Soc* 40/4:395-399 (1989)
61. Yagmahan B, Yenisey MM Ant colony optimization for multi-objective flow shop scheduling problem. *Comp Ind Eng* 54:411–420 (2008)

Tables

Table 1 Net set of non-dominated solutions obtained from various multi-objective flow shop scheduling algorithms for the benchmark problems given by [92], size (50×20).

Problem	01	Problem	02	Problem	03	Problem	04	Problem	05					
C_{max}	ΣC_i	C_{max}	ΣC_i	C_{max}	ΣC_i	C_{max}	ΣC_i	C_{max}	ΣC_i					
3893	136151	01	3742	128765	09	3696	125378	02	3769	128724	09	3660	125529	02
3894	136139	01	3744	128585	09	3697	123829	08	3772	128010	02	3678	124871	02
3896	135143	02	3745	127960	09	3700	122933	12	3775	127823	01	3682	124802	02
3899	134831	02	3747	127168	09	3716	122915	01	3792	127464	06	3687	124401	02
3900	133536	02	3754	126717	01	3724	122798	01	3807	127401	11	3735	124062	05
3902	133514	02	3756	126689	01	3738	122741	01	3810	127165	12	3746	124061	05
3904	133429	02	3765	126320	01	3746	120940	03	3811	127062	12	3747	123698	02
3912	133422	12	3777	126223	12	3766	120826	03	3816	127055	12	3761	123222	02
3913	133178	12	3800	125940	03	3768	120822	03	3850	126443	04	3767	121800	02
3915	133172	12	3840	125783	03	3782	120684	03	3851	126357	04	3786	121568	02
3922	133140	02	3847	125781	03	3792	120654	03	3853	126311	04	3862	121357	02
3953	131506	04	3854	124417	03	3795	120321	04	3860	125953	04	3864	121241	02
3954	131481	04	3860	124336	03	3804	119927	05	3863	125008	05	3868	121203	02
3958	131153	04	3864	123367	04	3810	119884	05	3874	124431	05	3875	121186	02
3997	130965	05	3869	123282	04	3812	119853	05	3886	124197	05	3878	121172	02
3998	130351	05	3872	123049	04	3814	119842	05	3888	124149	05	3887	121079	02
4018	130311	05	3875	123025	04	3817	119150	05	3900	123599	04	3902	121047	04
4022	130283	05	3925	122972	03	3818	119050	05	3910	123524	04	3905	121014	04
4030	130076	05	3930	122878	03	3822	118978	05	3982	123506	04	3912	120901	04
4031	129835	05	3950	122766	05	3824	118925	05	3984	123247	04	3927	120624	02
4036	129807	05	3954	122663	05	3825	118894	05	3985	123211	04	3959	120593	02
4049	129451	05	3964	122477	05	3842	118880	05	3993	123102	04	3974	120577	02

4068	129436	05	3968	122430	05	3853	118777	05	4019	123010	04	4022	120475	06
4147	127004	01	3983	122374	05	3859	118761	05	4072	122959	04	-	-	-
4220	126975	01	3986	122366	05	3879	118731	05	4076	122904	04	-	-	-
-	-	-	4020	122309	05	3882	118628	05	4127	122886	04	-	-	-
-	-	-	4060	121870	04	-	-	-	4150	122847	04	-	-	-
-	-	-	4072	121708	04	-	-	-	-	-	-	-	-	-
-	-	-	4079	121556	04	-	-	-	-	-	-	-	-	-
-	-	-	4088	121443	04	-	-	-	-	-	-	-	-	-
-	-	-	4099	121383	04	-	-	-	-	-	-	-	-	-
-	-	-	4104	121275	04	-	-	-	-	-	-	-	-	-
-	-	-	4109	121175	04	-	-	-	-	-	-	-	-	-

continued on next page

For each problem: 1st column indicates makespan; 2nd column indicates total flow time; 3rd column indicates the algorithm that yielded the corresponding solution as follows: 01 is MOSA1; 02 is MOSAD; 03 is MOSA1; 04 is MOSA2; 05 is PGA-ALS; 06 is IB8S; 07 is DB4S; 08 is IB4S; 09 is JB4S; 10 is NB4S; 11 is DB2A; and 12 is JB2A.

Table 1 Net set of non-dominated solutions obtained from various multi-objective flow shop scheduling algorithms for the benchmark problems given by [92], size (50×20) (continued).

Problem 6			Problem 7			Problem 8			Problem 9			Problem 10		
C_{max}	ΣC_i		C_{max}	ΣC_i		C_{max}	ΣC_i		C_{max}	ΣC_i		C_{max}	ΣC_i	
3724	130563	10	3763	131057	10	3766	130293	02	3786	133708	01	3802	130071	02
3725	130556	10	3767	131054	10	3767	130241	02	3788	133606	01	3804	129928	02
3727	130497	10	3770	131049	10	3769	130195	02	3791	132809	01	3819	129919	02
3731	130480	10	3773	130828	10	3770	130036	02	3797	132723	01	3855	128334	05
3734	130231	10	3778	130658	10	3771	129637	02	3808	130909	02	3869	128130	05
3736	130080	07	3781	130482	01	3773	129632	02	3816	130865	02	3896	127440	03
3739	129647	07	3784	130468	01	3774	129623	02	3817	130787	02	3907	127398	03
3740	129176	08	3787	130320	01	3776	129316	02	3818	130465	02	3914	126755	04
3743	128707	08	3789	129999	01	3785	128762	01	3822	130426	02	3915	126697	04
3745	128480	08	3790	129973	01	3808	128629	04	3830	130069	04	3918	126694	04
3746	128471	07	3791	129922	01	3816	128484	04	3833	129452	04	3922	126618	04
3751	127739	02	3792	129654	01	3817	128451	04	3848	129432	04	3936	126462	04
3790	127698	05	3804	129612	03	3819	127629	04	3862	129262	04	3952	126423	04
3791	126759	05	3808	129272	01	3823	127583	04	3864	128145	05	4046	126388	04
3807	126439	05	3813	129244	01	3836	127515	04	3872	128139	05	-	-	-
3810	126346	05	3818	128781	05	3854	127135	03	3879	127913	05	-	-	-
3814	126314	05	3823	128778	05	3860	126727	03	3880	127654	05	-	-	-
3821	125839	05	3829	128396	05	3865	125871	05	3883	127306	05	-	-	-
3827	125682	05	3833	128313	05	3869	125840	05	3896	127209	05	-	-	-
3834	125579	05	3837	128277	05	3878	125759	05	3905	127141	05	-	-	-
3836	125567	05	3840	127916	05	3884	125579	05	3916	126976	05	-	-	-
3840	125468	05	3841	127912	05	3889	125331	05	3917	126582	04	-	-	-
3841	125462	05	3854	127910	05	3891	125236	05	3921	126454	04	-	-	-

3843	125438	05	3867	12695	05	3894	125203	05	3924	126150	05	-	-	-
3845	125422	05	3892	126540	05	3901	125079	05	3933	125771	04	-	-	-
3848	124824	04	3894	126455	05	3905	125063	05	3954	125619	05	-	-	-
3862	124633	04	3902	126336	05	3910	125024	05	3965	125355	04	-	-	-
3878	124553	04	3927	126311	05	3918	124859	05	3974	125334	04	-	-	-
3886	124538	04	3953	126063	03	3929	124794	05	3981	125192	04	-	-	-
3891	123973	04	3954	126045	03	3931	124760	05	3982	125187	04	-	-	-
3933	123824	04	4032	125949	03	3935	124742	05	3983	125142	04	-	-	-
3944	123793	04	4098	125884	05	4076	124696	05	3984	125137	04	-	-	-
3946	123766	04	4145	125853	07	4127	124677	05	3988	125111	04	-	-	-
3956	123683	04	4147	125852	07	4131	124539	07	3990	125031	04	-	-	-
3959	123601	04	4269	125835	11	4161	124529	07	3991	125026	04	-	-	-
3964	123592	04	4269	125835	11	4176	124518	07	4004	124874	04	-	-	-
3979	123535	04	4273	125827	11	-	-	-	4006	124857	04	-	-	-
4008	123163	04	4273	125827	11	-	-	-	4013	124701	04	-	-	-
4011	123121	04	-	-	-	-	-	-	4040	124618	04	-	-	-
4029	121996	01	-	-	-	-	-	-	4043	124467	04	-	-	-
-	-	-	-	-	-	-	-	-	4044	124385	04	-	-	-
-	-	-	-	-	-	-	-	-	4052	124326	04	-	-	-
-	-	-	-	-	-	-	-	-	4056	124317	04	-	-	-
-	-	-	-	-	-	-	-	-	4067	123216	01	-	-	-
-	-	-	-	-	-	-	-	-	4074	123163	01	-	-	-

For each problem: 1st column indicates makespan; 2nd column indicates total flow time; 3rd column indicates the algorithm that yielded the corresponding solution as follows: 01 is MOSA1; 02 is MOSAD; 03 is MOSA1; 04 is MOSA2; 05 is PGA-ALS; 06 is IB8S; 07 is DB4S; 08 is IB4S; 09 is JB4S; 10 is NB4S; 11 is DB2A; and 12 is JB2A.

Table 2 Net set of non-dominated solutions obtained from various multi-objective flow shop scheduling algorithms for the benchmark problems given by [92], size (20×20).

Problem 1			Problem 2			Problem 3			Problem 4			Problem 5		
C_{max}	ΣC_i		C_{max}	ΣC_i		C_{max}	ΣC_i		C_{max}	ΣC_i		C_{max}	ΣC_i	
2246	36004	01	2101	33147	03	2239	35272	01	2166	32278	01	2271	34985	01
2248	35684	01	2105	33130	03	2253	35202	01	2167	32257	01	2278	34977	01
2256	35352	01	2108	32905	06	2258	34920	01	2171	32146	01	2450	34843	03
2298	35163	01	2109	32902	03	2440	34664	04	2249	32124	03	2521	34787	03
2317	35117	01	2110	32810	03	2452	34530	04	2251	32121	03	2533	34727	03
2334	35065	01	2117	32802	03	2463	34423	03	2256	32095	03	2548	34706	03
2344	34954	02	2119	32801	03	2465	34377	03	2258	32039	03	2560	34659	02
2351	34533	02	2122	32782	03	2466	34364	03	2260	31993	03	2602	34638	06
2352	34467	02	2132	32771	02	2481	34362	03	2261	31928	03	2617	34590	06
2355	34374	02	2134	32755	03	2483	34303	04	2263	31855	03	-	-	-
2388	34026	03	2135	32499	03	2484	34127	02	2264	31826	03,07	-	-	-
2402	33911	02,03	2145	32482	02	2547	34101	02,06	2265	31804	02,03,04	-	-	-
2406	33866	07	2173	32385	02	2549	34084	02	2276	31753	04	-	-	-
2410	33861	07	2182	32165	03	2555	34072	06,10	2289	31743	02,03,04	-	-	-
2412	33827	02,03,05,08	2195	32067	03	2564	34051	02,03	2301	31708	04	-	-	-
2418	33799	02,03	2198	32062	02	2567	34016	02,03	2311	31690	04	-	-	-
2427	33742	02	2200	31959	02	2578	33977	02	2387	31677	04	-	-	-
2437	33623	02	2205	31953	03	2579	33932	02,10	-	-	-	-	-	-
-	-	-	2218	31924	03	2608	33920	02,07	-	-	-	-	-	-
-	-	-	2221	31870	03	-	-	-	-	-	-	-	-	-
-	-	-	2249	31760	04	-	-	-	-	-	-	-	-	-
-	-	-	2261	31745	03	-	-	-	-	-	-	-	-	-
-	-	-	2264	31738	02	-	-	-	-	-	-	-	-	-
-	-	-	2266	31675	02	-	-	-	-	-	-	-	-	-

-	-	2286	31670	02	-	-	-	-	-	-
-	-	2307	31669	03	-	-	-	-	-	-
-	-	2318	31603	03	-	-	-	-	-	-
-	-	2334	31587	03	-	-	-	-	-	continued on next page

For each problem: 1st column indicates makespan; 2nd column indicates total flow time; 3rd column indicates the algorithm that yielded the corresponding solution as follows: 01 is MOSA1; 02 is MOSA1; 03 is MOSA2; 04 is MOGLS; 05 is IB8S; 06 is CB4S; 07 is IB4S; 08 is NB4S; 09 is IB2S; and 10 is DB4S.

Table 2 Net set of non-dominated solutions obtained from various multi-objective flow shop scheduling algorithms for the benchmark problems given by [92], size (20×20) (continued).

Problem 6			Problem 7			Problem 8			Problem 9			Problem 10		
C_{max}	$\sum C_i$		C_{max}	$\sum C_i$		C_{max}	$\sum C_i$		C_{max}	$\sum C_i$		C_{max}	$\sum C_i$	
2202	33516	01	2228	34123	01	2165	33700	01	2242	34966	03	2169	33155	01
2204	33413	01	2234	34088	01	2166	33282	01	2243	34884	03	2196	33125	02
2216	33311	01	2239	34026	01	2271	33280	03	2245	34810	03	2222	33092	03
2266	33264	09	2240	33969	01	2291	33263	02	2248	34363	03	2229	32961	03
2269	33171	02	2257	33645	01	2293	33247	03	2261	34344	03	2230	32901	03
2271	32879	02	2283	33482	02	2294	33246	02	2263	34292	02	2231	32864	03
2284	32778	04	2305	33390	02,03	2301	33237	03	2285	34279	03	2244	32735	02
2315	32754	04	2317	33361	03	2302	33235	03	2295	34200	04	2258	32719	02
2318	32722	04	2320	33325	03	2305	33130	02	2301	34190	04	2263	32646	02,03
2322	32643	03	2324	33295	02,03	2312	33108	03	2308	34176	04	2271	32624	03
2365	32616	03	2336	33253	02	2316	33004	03	2315	34093	04	2272	32623	03
2370	32612	03	2347	33244	02	2317	32902	02	2332	34090	02	2273	32526	03
2372	32564	06	2351	33212	03	2323	32875	02	2338	34088	03	2274	32506	02,04
-	-	-	2368	33204	02	2339	32867	03	2343	33959	02	2283	32461	01
-	-	-	2378	33190	02	2358	32848	02	2372	33923	03	2289	32441	02
-	-	-	2386	33184	02	2359	32808	02	2379	33772	03	2298	32434	02,04
-	-	-	2389	33083	05	2360	32775	03	2418	33734	03	2307	32418	01
-	-	-	-	-	-	2377	32743	03	2419	33729	03	2309	32331	02,03,07
-	-	-	-	-	-	2380	32663	03,04	2427	33722	03,05	2338	32299	02
-	-	-	-	-	-	2394	32655	03,04	2429	33715	03	2339	32292	03
-	-	-	-	-	-	2403	32590	05	2434	33713	03,05	2345	32269	02
-	-	-	-	-	-	2406	32565	05	2441	33669	02,03	-	-	-
-	-	-	-	-	-	2415	32509	05	2479	33644	03,05	-	-	-

-	-	-	-	-	2417	32467	01	2516	33639	03;05	-	-	-
-	-	-	-	-	2458	32455	05	-	-	-	-	-	-
-	-	-	-	-	2492	32444	01;03;04;08	-	-	-	-	-	-
-	-	-	-	-	2524	32412	05	-	-	-	-	-	-

For each problem: 1st column indicates makespan, 2nd column indicates total flow time, 3rd column indicates the algorithm that yielded the corresponding solution as follows: 01 is MOSA1; 02 is MOSA1; 03 is MOSA2; 04 is MOGLS; 05 is IB8S; 06 is CB4S; 07 is IB4S; 08 is NB4S; 09 is IB2S; and 10 is DB4S.

Table 3 Net set of non-dominated solutions obtained from various multi-objective flow shop scheduling algorithms for the benchmark problems given by [92], size (100×20).

Problem	1	Problem 2		Problem 3		Problem 4		Problem 5						
C_{max}	ΣC_i	C_{max}	ΣC_i	C_{max}	ΣC_i	C_{max}	ΣC_i	C_{max}	ΣC_i					
6307	402146	09	6256	408911	05	6330	404099	11	6351	407159	09	6484	396337	04
6312	401153	09	6257	407339	05	6331	404091	11	6352	405852	09	6489	394165	04
6330	399770	07	6258	407241	05	6332	403274	11	6366	405150	01	6512	393698	04
6332	399640	07	6259	407121	05	6334	402931	11	6400	404989	10	6516	392699	04
6334	399472	07	6265	407075	05	6336	402542	11	6403	404316	10	6521	390885	04
6345	396719	07	6268	406208	05	6341	401393	02	6405	404236	10	6526	390659	04
6352	396549	01	6270	406162	05	6343	401101	02	6422	401301	04	6571	387708	04
6353	396315	01	6271	405559	05	6345	400431	02	6464	400242	04	6601	387400	04
6358	396270	01	6272	404604	05	6346	399420	02	6466	398617	04	6607	387387	04
6360	395948	01	6275	404009	05	6350	399220	02	6504	398507	03	6610	386987	04
6362	394871	01	6277	400329	08	6352	399205	02	6506	395319	04	6613	386425	04
6430	394617	01	6281	399999	05	6356	398870	02	6509	394379	04	6627	386045	04
6431	394270	01	6282	399589	08	6365	398151	05	6512	393968	04	6633	384976	04
6455	394159	01	6284	399354	08	6370	397432	05	6528	393958	04	6637	384906	04
6464	393423	11	6286	397790	08	6495	396715	04	6567	388774	04	6640	384853	04
6467	393330	11	6287	397380	08	6498	396651	04	6569	388316	04	6649	384822	04
6468	393027	11	6289	397074	08	6501	394936	04	6572	388248	04	6654	383531	03
6469	392981	11	6290	396941	08	6507	394916	04	6573	386616	04	6662	380537	04
6470	392103	04	6397	396828	07	6508	394763	03	6579	384987	04	6680	380465	04
6471	391700	04	6434	395376	03	6509	393612	04	6582	384860	04	6712	380034	04
6507	391673	04	6436	395031	03	6522	392238	04	6599	384025	04	6713	379528	04
6511	390320	03	6438	394799	03	6535	392125	04	6614	383895	04	6714	379452	04
6529	389635	04	6440	394636	03	6543	391958	04	6656	383841	04	6735	379371	04

6537	389572	04	6447	393778	03	6544	391888	04	6689	383773	04	6770	379093	04
6541	388971	03	6485	389578	03	6552	391632	04	6691	383721	04	6819	378985	04
6546	388962	11	6487	389053	03	6558	391136	04	6693	383629	04	6843	378783	04
6549	387820	03	6499	387616	03	6560	390630	04	6705	383512	04	6851	378768	04
6551	385700	03	6507	387549	03	6573	389211	04	6739	383380	04	6862	378653	04
6565	385312	04	6580	387278	03	6574	389209	04	6741	383319	04	-	-	-
6569	384713	03	6581	387219	03	6578	388946	04	6748	383300	04	-	-	-
6570	383196	04	6606	386521	03	6586	388751	04	6749	383253	04	-	-	-
6587	383184	04	6610	386378	03	6588	388716	04	6829	382998	04	-	-	-
6588	382733	04	6662	385809	04	6619	388478	04	6861	382981	04	-	-	-
6590	382380	04	6666	385302	04	6620	386668	04	-	-	-	-	-	-
6597	382285	04	6694	385181	04	6637	386617	04	-	-	-	-	-	-
6605	382150	03	6695	385176	04	6646	386314	04	-	-	-	-	-	-
6620	381682	03	6698	384809	04	6647	386251	04	-	-	-	-	-	-
6630	381490	04	6699	384129	03	6653	386180	04	-	-	-	-	-	-
6639	381452	04	6725	383997	03	6696	385723	04	-	-	-	-	-	-
6657	381387	03	6801	383977	03	6724	385279	04	-	-	-	-	-	-
6666	379873	04	-	-	-	6740	385096	03	-	-	-	-	-	-
6668	379765	04	-	-	-	6741	384603	04	-	-	-	-	-	-
6670	379409	04	-	-	-	6766	384495	04	-	-	-	-	-	-
6674	377697	01	-	-	-	6782	384027	04	-	-	-	-	-	-
6691	376944	01	-	-	-	6796	383962	04	-	-	-	-	-	-
-	-	-	-	-	-	6801	383632	04	-	-	-	-	-	-
-	-	-	-	-	-	6817	380510	09	-	-	-	-	-	-
-	-	-	-	-	-	6830	380089	09	-	-	-	-	-	-
-	-	-	-	-	-	6860	380075	09	-	-	-	-	-	-

on next page

For each problem: 1st column indicates makespan, 2nd column indicates total flow time, 3rd column indicates the algorithm that yielded the corresponding solution as follows: 01 is MOSA1; 02 is MOSAD, 03 is MOSA1; 04 is MOSA2; 05 is IB8S; 06 is CB4S; 07 is DB4S; 08 is JB4S; 09 is NB4S; 10 is DB2A; 11 is JB2A; and 12 is IB2S.

Table 3 Net set of non-dominated solutions obtained from various multi-objective flow shop scheduling algorithms for the benchmark problems given by [92], size (100×20) (continued).

Problem 6			Problem 7			Problem 8			Problem 9			Problem 10		
C_{max}	ΣC_i		C_{max}	ΣC_i		C_{max}	ΣC_i		C_{max}	ΣC_i		C_{max}	ΣC_i	
6450	401865	08	6318	410310	10	6500	417356	07	6407	404406	10	6542	416348	05
6459	401313	08	6319	409977	01	6501	415737	07	6408	404400	10	6546	415297	02
6467	400679	08	6324	409932	01	6511	415297	08	6410	404269	10	6547	414551	02
6468	400620	08	6326	405062	11	6512	413906	08	6411	403485	10	6558	414341	02
6487	399744	06	6329	404984	11	6516	412770	08	6413	402363	07	6570	413380	10
6492	399741	06	6331	404891	11	6540	412707	09	6421	402358	07	6573	413156	10
6494	399633	06	6338	403414	10	6541	411407	06	6426	400738	10	6578	412937	05
6506	399468	02	6362	403100	06	6605	411120	02	6477	399756	04	6598	411116	05
6512	399148	05	6363	403077	06	6637	410851	12	6485	398654	04	6632	409168	04
6513	399126	05	6368	402695	06	6638	410415	04	6490	397105	04	6633	406828	04
6514	399095	02	6394	402622	08	6641	410368	04	6491	397006	04	6634	406169	04
6517	396836	04	6398	402266	08	6645	409553	04	6495	397005	04	6676	403143	04
6518	395278	04	6418	400814	06	6665	409442	03	6496	396908	04	6679	401929	04
6557	394615	10	6434	400237	07.09	6672	408679	03	6544	396494	04	6681	400989	04
6559	393663	10	6435	400148	07.09	6680	408611	03	6548	395375	04	6716	400173	04
6613	383754	04	6448	400087	07.09	6710	408515	03	6552	394721	04	6721	399777	04
6619	383713	04	6463	397143	04	6719	405961	04	6558	394679	04	6746	399027	04
6655	383473	04	6468	397056	04	6727	405825	04	6559	394455	04	6762	398974	04
6663	382823	04	6471	396646	04	6744	399739	04	6570	393173	04	6768	395677	03
6736	382413	04	6472	396578	04	6779	399369	04	6579	392942	04	6789	395650	03
6774	382185	04	6473	395124	04	6809	399326	04	6588	391050	04	6792	391317	04
6845	382069	04	6566	393835	04	6826	398840	03	6595	390779	04	6793	390985	04
6871	381966	04	6572	393817	04	6830	398536	03	6614	389451	04	6795	390911	04

6879	381916	04	6580	391542	04	6834	397808	04	6640	389203	04	6811	390531	04
-	-	-	6605	389318	04	6847	397782	04	6649	389151	04	6855	390493	04
-	-	-	6612	387976	04	6858	397074	04	6652	388884	04	6868	390265	04
-	-	-	6621	387460	04	6860	396013	04	6734	386833	04	6902	390177	04
-	-	-	6691	387074	04	6890	396006	04	6735	386263	04	6910	390113	04
-	-	-	6694	386868	04	6894	395922	04	6742	386125	04	6991	390061	04
-	-	-	6720	386514	04	6905	395678	04	6798	386097	04	7043	390059	04
-	-	-	6722	386511	04	6921	395003	04	6813	386073	04	-	-	-
-	-	-	6748	385668	04	6980	394745	04	6815	386055	04	-	-	-
-	-	-	6752	385587	04	6984	394535	04	6830	386045	04	-	-	-
-	-	-	6758	385529	04	6985	394447	04	-	-	-	-	-	-
-	-	-	6760	385427	04	-	-	-	-	-	-	-	-	-

For each problem: 1st column indicates makespan; 2nd column indicates total flow time; 3rd column indicates the algorithm that yielded the corresponding solution as follows: 01 is MOSA1; 02 is MOSAD; 03 is MOSA1; 04 is MOSA2; 05 is IB85; 06 is CB4S; 07 is DB4S; 08 is JB4S; 09 is NB4S; 10 is DB2A; 11 is JB2A; and 12 is IB2S.

Table 4. Number of sequences generated during the entire search process.

<i>n</i>	<i>m</i>	MOSAD	MOSAI	MOSA1	MOSA2	GPWGA	MOGLS	ENG	<i>a-posteriori</i>
20	05	15.312	19.666	-	-	-	-	-	-
20	10	23.801	28.032	-	-	-	-	-	-
20	20	21.187	31.429	294.500	562.020	152.280	152.280	152.280	2.310
50	05	23.427	30.994	-	-	-	-	-	-
50	10	41.349	40.177	-	-	-	-	-	-
50	20	39.722	56.558	768.320	1.458.240	434.700	434.700	434.700	14.025
100	05	28.212	34.319	-	-	-	-	-	-
100	10	35.392	47.111	-	-	-	-	-	-
100	20	59.653	58.278	1.582.020	2.975.940	899.400	899.700	899.700	55.550

A local approach to automated correction of violated precedence and resource constraints in manually altered schedules

Roman Barták • Tomáš Skalický

Abstract Users of automated scheduling systems frequently require an interactive approach to scheduling where they can manually modify the schedules. Because of complexity and cohesion of scheduling relations, it may happen that manual modification introduces flaws to the schedule, namely the altered schedule violates some constraints such as precedence relations or limited capacity of resources. It is useful to automatically correct these flaws while minimizing other required changes of the schedule. In this paper we suggest a fully automated approach to correcting violated precedence and unary resource constraints. The presented techniques attempt to alter minimally the existing schedule by doing the changes only locally in the area of the flaw.

1 Introduction

Fully automated scheduling seems like the Holy Grail of scheduling community, but most practitioners frequently require the freedom of manually altering the generated schedules, for example, to introduce some aspects of the particular area that were hard to formalize and hence are not reflected in the automatically generated schedule. Despite the high experience of human schedulers, there is a high probability that after a manual modification some flaws are introduced to the schedule. This probability is higher if the density of scheduling constraints is large and the constraints are highly coupled. For example, delaying one activity may delay other dependent activities due to precedence constraints between them or due to limited capacity of resources. It might be enough just to detect such violations and report them to the user who will be responsible for manual correction. Nevertheless, such manual corrections may be boring and sometimes very hard because of interconnectivity of the constraints (correction of one flaw introduces other flaws etc.). Hence we suggest a fully automated (“push button”) approach to correcting schedules after manual modification. Namely, we address the problem of correcting precedence constraints and unary resource constraints by shifting locally the affected activities in time.

Roman Barták
Charles University in Prague, Faculty of Mathematics and Physics
E-mail: bartak@ktiml.mff.cuni.cz

Tomáš Skalický
Charles University in Prague, Faculty of Mathematics and Physics
E-mail: skalicky.tomas@gmail.com

This paper studies the problem of correcting general schedules consisting of a set of non-interruptible activities with fixed durations; each activity is allocated to one or several unary resources (at most one activity can be processed at any time by a unary resource, but the activity may require more resources at the same time) and the activities are connected via precedence constraints (activity can start only after all its predecessors finished). The primary motivation for the research is providing an automated tool for correcting manually altered schedules in interactive Gantt chart environments (a “push button” approach). Nevertheless, the proposed repair techniques can also be used for example in the intensification stage of scheduling algorithms based on genetic algorithms or meta-heuristics. From another perspective, the proposed techniques belong to the group of re-scheduling algorithms.

We assume that the initial allocation of all activities to time (an initial schedule) is known. This time allocation may violate some precedence constraints (activity starts before some of its predecessors finishes) or some resource constraints (two or more activities are processed at the same time by the same resource). The goal is to correct the schedule (re-schedule) by shifting the activities in time, that is, to find a feasible schedule that does not violate any constraint. Moreover, the new schedule should not differ a lot from the initial time allocation of activities. Note that finding a feasible schedule is always possible unless there is a loop in the precedence constraints – activities can always be shifted to future as there are no deadlines. To minimize the number of changes between the initial and final schedule we apply a local approach, where particular flaws are repaired by local changes of affected activities rather than generating a completely new schedule from scratch. A local repair may introduce other flaws in the neighborhood which spread like a wave until all flaws are resolved. We use a three-step approach to repair a schedule. In the first step, loops of precedence constraints are detected and the user is asked to break each loop by removing at least one precedence constraint from it. In the second step, we repair all precedence constraints; two methods are suggested for this repair. Finally in the third step we repair violation of resource capacity constraints while keeping the precedence constraints valid. Each repair is realized by shifting affected activities locally in time.

The paper is organized as follows. First, a formal definition of the scheduling problem is given. Then we discuss some existing techniques that can be applied to schedule repair. After that, our three-stage approach to schedule repair is described in detail and proof of soundness is given. In conclusion we briefly discuss possible future steps.

2 Problem formulation

We use a fairly general description of the scheduling problem that fits especially fields such as construction and production scheduling. The Resource-Constrained Project Scheduling Problem (RCPSP) [3] is probably the closest classical scheduling problem though there are some differences as described below.

We assume a finite set of activities Act , each activity $A \in Act$ has a fixed duration d_A and it is non-interruptible (activity must run from its start till its end without interruption). Let s_A be the start time of activity A – the minimum start time of any activity is zero (schedule start), but there is no deadline. There is a set $Prec$ of precedence constraints between the activities in the form $(A \rightarrow B)$; A is called a predecessor of B and B is called a successor of A . Formally for each precedence relation $(A \rightarrow B) \in Prec$ the following constraint must hold:

$$s_A + d_A \leq s_B \quad (1)$$

Let Res be a finite set of unary resources, that is, each resource can process at most one activity at any time. For each activity $A \in Act$ there is a set of required resources $r(A) \subseteq Res$. Activity A requires all resources from the set $r(A)$ for processing, that is, A occupies each resource $R \in r(A)$ for the time period $[s_A, s_A + d_A]$. The resource constraints can be formally expressed in the following way:

$$\forall A, B \in Act \text{ s.t. } r(A) \cap r(B) \neq \emptyset: s_A + d_A \leq s_B \vee s_B + d_B \leq s_A \quad (2)$$

The above resource constraint says that two activities A and B sharing the same resource cannot overlap in time (either A precedes B or B precedes A).

A *schedule* is a particular allocation of activities to time, formally it is a mapping of all variables s_A to natural numbers N_0 (including zero). A *feasible schedule* is a schedule that satisfies constraints (1) and (2). Notice that resource allocation is not part of the problem (activities are already allocated to resources). It is easy to prove that a feasible schedule always exists provided that there is no loop in the precedence constraints (for example $A \rightarrow B \rightarrow C \rightarrow A$). It is possible to topologically order all activities respecting the precedence constraints (precedence constraints define the partial ordering of activities) and then to allocate activities in this order to earliest possible times while respecting the precedence (1) and resource (2) constraints (activity can always be shifted to future if resource is not available at some time).

We do not assume any particular objective function in our scheduling problem. Makespan is a typical objective for RCPSP and other scheduling problems and our techniques for schedule repair try not to extend the makespan a lot beyond the makespan of the initial schedule. Though keeping the schedule compact (with small makespan) we are not really optimizing makespan or any other objective function. One of the reasons is that real-life objectives are frequently different from makespan (for example on-time-in-full is required) and it is hard or even impossible to formally express the real objective. Hence, we use the assumption that the initial schedule has a good quality from the user point of view and so the repaired schedule should not differ a lot from the initial schedule. Nevertheless, minimizing the difference between the schedules is more the nature of the presented repair techniques (repairing flaws locally) than formal minimization of the difference between the schedules as for example studied in [5,2].

The problem that we are solving in this paper can be stated as follows: given some schedule S, find a feasible schedule S' that does not differ a lot from S. The difference between schedules S and S' can be formalized in the following way:

$$\text{difference}(S, S') = \sum_{A \in Act} |s_A - s'_A| \quad (3)$$

where s_A is the start time of activity A in S and s'_A is the start time of A in S'. Notice that the only way to modify the schedule is via changing values of variables s_A . As mentioned above, we are not strictly minimizing the objective (3), we are trying to achieve a good value of $\text{difference}(S, S')$ by changing the values of s_A as little as possible (locally) to repair a violated constraint.

3 Related works

Dynamic scheduling is not a new area and though most scheduling research still focuses on static problems that do not change over time, there is an enlarging interest in studying dynamic aspects of scheduling [6]. This is mainly due to real-life scheduling problems that are primarily dynamic – machines break down, deliveries are delayed, workers become ill etc. As mentioned in the introduction, our motivation is slightly different and it goes from the area of interactive scheduling, where we need to repair manually modified schedules to satisfy all constraints. Nevertheless, the used technology is very similar.

The simplest approach to re-scheduling is generating a new schedule from scratch, also called total re-scheduling. This is the best technique when there are many disruptions in the schedule, but it has the disadvantage of generating a schedule completely different from the original schedule. We focus more on local repairs of the schedule with the motivation to keep the schedule similar to the original schedule. The method of generating a new schedule from scratch can be “localized” by removing some elements from the schedule and then adding them back without violation of constraints. Iterative Flattening Search [8] is an example of such a method where in the relaxation step all violated (and some other) constraints are removed from the problem (typically decisions about the ordering of activities) and then in the flattening stage the possible conflicts in the schedule are resolved by adding the constraints

back. Naturally, the question is how to relax the schedule to be able to resolve all possible conflicts (in the extreme case, all decision constraints are removed). Iterative Forward Search [7] uses a similar approach but it removes activities participating in violated constraints and then schedules these activities again. It has been proposed originally for timetabling problems with limited temporal constraints but for scheduling problems with temporally connected activities it may require re-scheduling many activities as it works with partial consistent schedules. Nevertheless, thanks to iterative improvements of schedules, this approach has been successfully applied to minimal perturbation problems (MPP) introduced in [5] and redefined in [2]. MPP focuses on finding a solution to a modified problem with minimal differences (perturbations) from the solution of the original problem. Hence, it is a primarily optimization problem with a specific objective function defined by the original solution. MPP can be solved from scratch like other optimization problems, for example using search techniques [2].

The approach studied in this paper belongs to heuristic-based repair algorithms where right shift rescheduling and affected operation rescheduling are the most frequently used techniques. *Right shift rescheduling* [4] repair is performed by globally shifting all remaining activities forward in time by the amount of disruption time. This introduces a gap in the schedule and it is not really appropriate for our type of problem. *Affected operations rescheduling* [1] reschedules only the activities that are directly or indirectly affected by the disruption. This heuristic was proposed to repair machine breakdowns but its generalization called modified Affected Operations Rescheduling [9] has been proposed to repair other typical disruptions seen in a job shop. We follow the idea of affected operations rescheduling but rather than assuming specific repair rules and repair operations such as insertion or deletion of activity, we allow only shifting the existing activity in time both forward and backward. Moreover, our techniques are designed to repair any number of constraint violations in the schedule. Recall that we are repairing manually modified schedules with flaws rather than reacting to instant disruptions from the job shop. Hence there might be more flaws spread in the schedule and we need to repair all of them while keeping the schedule as similar as possible to the original schedule.

4 Re-scheduling (repair) algorithm

As we already mentioned, we assume a typical scenario, where the human scheduler modifies an automatically generated schedule to reflect better the peculiarities of particular environment. The modification can affect any part of the scheduling problem introduced above – it is possible to change duration of activities, their position in time and required resources, to add or delete precedence constraints or even to add or delete activities and resources (in case of changing the set of activities, it is necessary to introduce a different measure of schedule difference, see for example [2]). By these modifications, it is quite easy to obtain an infeasible schedule where some of precedence or resource constraints are violated (we call the violated constraint a *flaw*). Though it is easy to detect and visualize the violated constraints (see Figure 1), it is frequently more complicated to repair them without introducing other flaws.

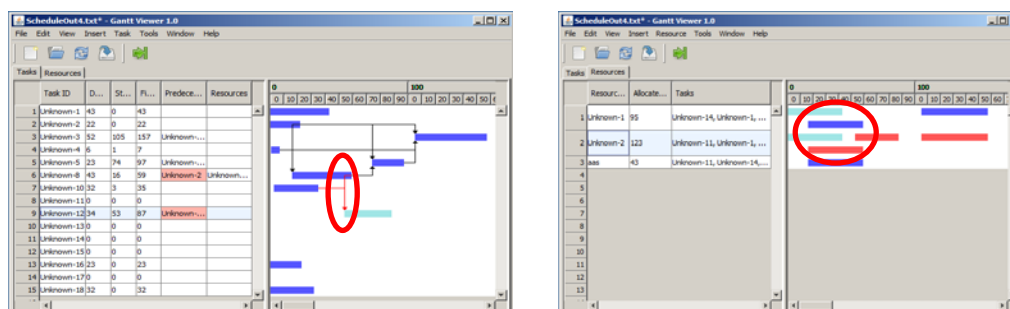


Fig. 1. Gantt charts visualization of violated precedence (left) and resource (right) constraint.

We suggest a schedule-repair method that mimics the behavior of a human scheduler by repairing flaws via local changes of time allocation of activities participating in the flaw. Naturally, this may introduce other flaws which need to be repaired and hence a systematic approach is necessary to prevent an infinite number of repairs (repairing one flaw introduces another flaw whose repair brings back the original flaw etc.). While such a systematic approach may be boring for a human, it is easy for a computer. The suggested method consists of three stages:

- detecting and breaking loops of precedence constraints,
- repairing violated precedence constraints,
- repairing violated resource constraints.

By modifying the set of precedence constraints, the user may unwittingly introduce a cycle between activities which prevents existence of the feasible schedule. Hence the first stage is detecting such loops and asking the user to remove some precedence constraint from each such loop. This is the only stage where user intervention is necessary¹; the other two repair stages are fully automated. Recall that if there are no loops of precedence constraints then a feasible schedule always exists. In the following sections we will describe each stage in more details.

4.1 Loop detection

We represent the scheduling problem as a directed graph $G = (E, V)$, where the set V of nodes equals the set Act of activities and there is edge (A, B) in E if and only if $(A \rightarrow B) \in Prec$. There exist standard methods for finding cycles in graphs and we adopted one of them. The method is based on repeating the following three steps until an empty graph is obtained:

1. repeatedly delete all nodes N from the graph such that there is either no incoming edge $(X, N) \in E$ or no outgoing edge $(N, X) \in E$; after this step, each remaining node in the graph has at least one predecessor and one successor
2. select any node from the graph (we use the node with the largest number of successors) and find a loop by depth-first search going in the direction of edges
3. present the loop to the user (in terms of activities) and remove the edge(s) suggested by the user (the particular precedence is also removed from $Prec$).

It is easy to prove that the above method removes all loops from the precedence constraints. In particular, by DFS in step 2 we must find a loop because each node has a successor (after step 1) and there is a finite number of nodes so at some time some node must be visited for the second time so the path between the first and second visit forms a loop. As we stop with an empty graph, no loops remain in the graph.

4.2 Precedence repair techniques

The goal of the second stage of the repair algorithm is to remove violation of all precedence constraints (1). This is possible for any schedule that does not contain loops in precedence relations which is exactly the schedule resulting from the first stage described in the previous section. We ignore violation of resource capacity constraints (2) at this stage.

The precedence $(A \rightarrow B) \in Prec$ is violated if $s_A + d_A > s_B$. The size of violation can be described by the following variable:

$$\text{diff}(A, B) = s_A + d_A - s_B.$$

¹ It is possible to randomly remove some precedence constraint from each loop or even to minimize the number of removed precedence constraints to break all loops, but in our opinion, the human decision is more appropriate.

Clearly, $\text{diff}(A,B)$ is positive if and only if precedence $(A \rightarrow B)$ is violated. To locally repair the violated precedence $(A \rightarrow B)$ we can shift A backward in time (decrease s_A) or shift B forward in time (increase s_B) or shift together A backward and B forward. Naturally, if we do not want to stretch the schedule (increase makespan) then decreasing s_A as much as possible (but no more than constraint (1) requires) is the preferred way of repair. To find out how much time is available for shifting A backward we introduce the following variable:

$$\begin{aligned} \text{freeOnLeft}(A) &= s_A && \text{if } A \text{ has no predecessors} \\ &= s_A - (s_{lp(A)} + d_{lp(A)}) && \text{if } A \text{ has some predecessor and } lp(A) \text{ denotes} \\ &&& \text{the latest predecessor of } A \text{ in the schedule;} \\ &&& lp(A) = \text{argmax}_{C: (C \rightarrow A) \in \text{Prec}} (s_C + d_C). \end{aligned}$$

The straightforward technique of repairing a violated precedence constraint $(A \rightarrow B)$ is shifting A backward as much as possible and then shifting B forward if necessary. This can be formally described by the following assignments:

$$\begin{aligned} s_A &\leftarrow s_A - \min(\text{freeOnLeft}(A), \text{diff}(A,B)) \\ s_B &\leftarrow s_A + d_A \end{aligned}$$

Clearly, after making the suggested modification of start times s_A and s_B , the precedence relation $(A \rightarrow B)$ is satisfied ($s_B = s_A + d_A$). However, does it always realize the idea of shifting A backward? If all precedence constraints $(C \rightarrow A)$ are satisfied then $\text{freeOnLeft}(A) \geq 0$ because $s_A \geq 0$ (definition of s_A) and $\forall C \in \text{Act}$ s.t. $(C \rightarrow A) \in \text{Prec}$: $s_A \geq s_C + d_C$ (according to (1)). However, if some $(C \rightarrow A)$ is violated then $\text{freeOnLeft}(A) < 0$ which actually means that A is shifted forward (s_A is increased) when repairing precedence $(A \rightarrow B)$. To prevent this unwanted behavior, it is enough to ensure that all precedence constraints $(C \rightarrow A)$ are satisfied before repairing the precedence constraint $(A \rightarrow B)$. Moreover, in such a case all precedence constraints $(C \rightarrow A)$ still remain valid after repairing $(A \rightarrow B)$. Hence, if we repair the violated precedence constraints in the right order, namely from left to right, then it is enough to repair each violation exactly once. Let us assign a unique index i to each precedence $(A \rightarrow B)$, denoted $(A \rightarrow B)_i$, in such a way that if we have two indexed precedence relations $(A \rightarrow B)_i$ and $(B \rightarrow C)_j$ then $i < j$. This can be easily realized by sorting first the activities according to the topological order satisfying the partial order defined by the precedence relations (this is always possible as there are no cycles) and then indexing the precedence relations according to this order (see Figure 2).

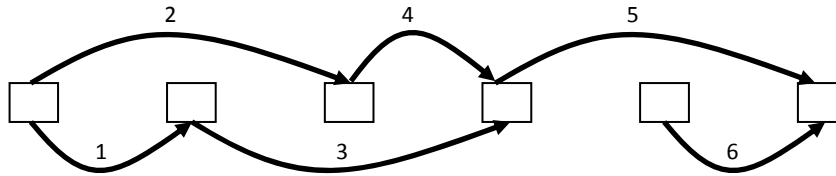


Fig. 2. Possible topological ordering of precedence constraints.

After defining the order of precedence relations the following pseudo-code *PrecRep* describes the repair algorithm:

```

algorithm PrecRep
1  while any precedence is violated do
2      select violated precedence  $(A \rightarrow B)_i$  such that  $i$  is minimal
3       $s_A \leftarrow s_A - \min(\text{freeOnLeft}(A), \text{diff}(A,B))$ 
4       $s_B \leftarrow s_A + d_A$ 
5  end while
end PrecRep

```

Proposition 1: Algorithm *PrecRep* is sound and complete with respect to producing a schedule without violation of precedence constraints.

Proof: Clearly, if the algorithm stops then there are no violated precedence constraints in the schedule (because of the condition in the while loop). Hence it is enough to show that the algorithm stops after a finite number of steps. In each iteration the algorithm repairs (at least) one precedence constraint. The precedence relations are repaired in the order of their indexes so when $(A \rightarrow B)_i$ is being repaired then all $(X \rightarrow Y)_j$ such that $j < i$ are satisfied (have already been repaired). Moreover, no such relation $(X \rightarrow Y)_j$ where $j < i$ is violated by the repair of $(A \rightarrow B)_i$. Note that only relations $(C \rightarrow A)$ and $(B \rightarrow D)$ are influenced by the repair of $(A \rightarrow B)_i$ because we can only shift A backward and B forward. Satisfaction of other precedence relations involving A or B, namely $(A \rightarrow C)$ and $(D \rightarrow B)$, is not influenced by the repair. As we already discussed, relations $(C \rightarrow A)$ are not violated by the repair. According to the ordering of precedence relations, relations $(B \rightarrow D)_k$ have larger index than $(A \rightarrow B)_i$ ($k > i$). In summary, after each iteration of the while loop we increase index k such that $\forall j \leq k$ $(X \rightarrow Y)_j$ is satisfied by at least one. Hence, after at most m iterations, where m is the number of precedence constraints in the schedule (the largest index), we repair all precedence relations. ■

Algorithm *PrecRep* represents a straightforward way of repairing precedence constraints. Unfortunately, it can shift activities forward more than necessary and hence it can increase makespan and make the schedule less compact. Though the algorithm can shift activity A backward when repairing $(A \rightarrow B)$, it can shift A at most as the latest predecessor $lp(A)$ of A allows (see the definition of *FreeOnLeft*). Hence $lp(A)$ may block shifting A backward even if there is time. In particular, it might be possible to shift $lp(A)$ backward as well and hence to increase the time available for A (see Figure 3). To improve this behavior, we suggest a modification of the repair algorithm called *PrecRep-2* that exploits better available time on the left of activity A by shifting it beyond the horizon defined by $lp(A)$.

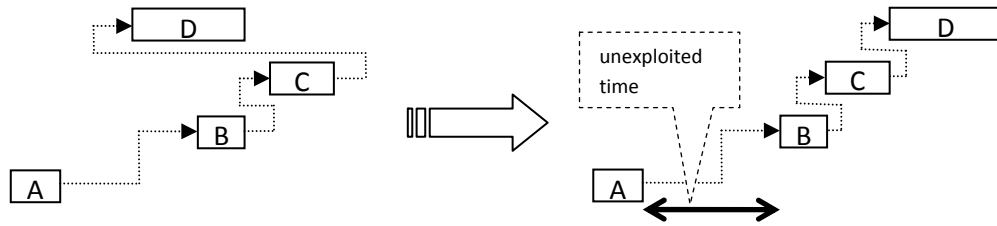


Fig. 3. Algorithm *PrecRep* does not exploit fully available time on left of D.

The idea of *PrecRep-2* algorithm is to shift A backward similarly to *PrecRep*, but if this is not enough to satisfy the constraint $(A \rightarrow B)$ ($\text{diff}(A,B)$ is still positive) then we shift A backward slightly more, in particular by $\text{truncate}(\text{diff}(A,B)/2)$, where $\text{truncate}(X)$ is the closest integer between X and 0, for example $\text{truncate}(3.7) = 3$. This way, we violate the constraint $(lp(A) \rightarrow A)$ which can be repaired later by shifting $lp(A)$ backward and so on. By this process, we can exploit better available time by shrinking the schedule. We only ensure that we do not violate the constraint $0 \leq s_A$ so the schedule does not stretch beyond the schedule start.

```

algorithm PrecRep-2
1   while any precedence is violated do
2       select violated precedence  $(A \rightarrow B)_i$  such that  $i$  is minimal
3        $s_A \leftarrow s_A - \min(\text{freeOnLeft}(A), \text{diff}(A,B))$ 
4        $s_A \leftarrow \max(0, s_A - \text{truncate}(\text{diff}(A,B)/2))$ 
5        $s_B \leftarrow s_A + d_A$ 
6   end while
end PrecRep-2
    
```

Lemma 1: If algorithm *PrecRep-2* repairs precedence $(A \rightarrow B)_i$ then after a finite number of iterations, the algorithm reaches a situation when all $(X \rightarrow Y)_j$ such that $j \leq i$ are satisfied. Moreover, the final start time s_A is not greater than the start time of A before repair of $(A \rightarrow B)$.

Proof: We shall prove the lemma by induction on the index of the repaired precedence. For $i = 1$ the lemma trivially holds, because the precedence $(A \rightarrow B)_1$ is repaired by the algorithm and s_A is not increased. When the algorithm started the repair of precedence $(A \rightarrow B)_i$, $i > 1$, all $(X \rightarrow Y)_j$ such that $j < i$ were satisfied. If these precedence relations are still satisfied after the repair then the lemma holds.

Assume that some precedence $(C \rightarrow A)$ has been violated by the backward shift of A . This may happen only after the assignment at line 4 so let old_s_A be the value of s_A before processing line 4 and new_s_A be the value of s_A after processing line 4. Note that if A starts at old_s_A then no precedence constraint $(C \rightarrow A)$ is violated. Let us take the violated precedence constraint $(C \rightarrow A)_i$ with the smallest index i that will be repaired next. According to the induction assumption, it is possible to repair this constraint and the value of s_C does not increase. Hence the new value of s_A is not greater than old_s_A because old_s_A satisfied the constraint $(C \rightarrow A)$ and s_C did not increase above its original value. In the same way, we can repair all violated precedence constraints $(C \rightarrow A)$ to reach the situation when all $(X \rightarrow Y)_j$ such that $j < i$ are satisfied again. The final value of s_A will not be greater than old_s_A .

Unfortunately, the constraint $(A \rightarrow B)_i$ may be violated again (when s_A increases) and we need to repair it. Nevertheless, one should realize that $diff(A, B)$ is now strictly smaller than it was before we repaired the constraint $(A \rightarrow B)$ for the first time. Note that when A starts at new_s_A then the constraint $(A \rightarrow B)$ is satisfied – we repaired it this way. However, after repairing all $(C \rightarrow A)$, the value of s_A might increase as much as to old_s_A ($new\_s_A < old\_s_A$) so the new $diff(A, B) \leq old\_s_A - new\_s_A$. From line 4 of the algorithm, we can see that new $diff(A, B)$ is at most half of the original $diff(A, B)$. Anyway, we need to repeat the above process again and next time $diff(A, B)$ will be even smaller. In the worst case, we stop when $diff(A, B) = 1$ because then the assignment at line 4 does not change s_A and hence no constraint $(C \rightarrow A)$ is violated so we reached the situation when all $(X \rightarrow Y)_j$ such that $j \leq i$ are satisfied. Moreover, we can see that s_A never went above old_s_A , which is not larger than the original value of s_A (old_s_A may be smaller than s_A because s_A might be decreased at line 3). ■

Proposition 2: Algorithm *PrecRep-2* is sound and complete with respect to producing a schedule without violation of precedence constraints.

Proof: If the algorithm stops then there are no violated precedence constraints in the schedule (because of the condition in the while loop) so the algorithm is sound. To prove completeness, it is necessary to show that the algorithm stops after a finite number of iterations (recall that a feasible schedule always exists). The precedence relations are repaired in the order of their indexes. According to lemma 1 after $(A \rightarrow B)_i$ is repaired then in a finite number of iterations the algorithm reaches a situation when all $(X \rightarrow Y)_j$ such that $j \leq i$ are satisfied. Hence, in the next step we will be repairing some precedence $(C \rightarrow D)_k$ where $k > i$ (if any such violated precedence still exists). Again, according to lemma 1 we can reach the situation when all $(X \rightarrow Y)_j$ such that $j \leq k$ are satisfied. We can continue this way until we reach the last index m . In summary, after a finite number of iterations the algorithm repairs all precedence constraints – the algorithm reaches a situation when all $(X \rightarrow Y)_j$ such that $j \leq m$ are satisfied. ■

Algorithm *PrecRep-2* exploits better available time (see Figure 4) but it is slower than *PrecRep* due to repeated “shrink-and-stretch” stages after violating the already repaired constraints. The open question is if the time complexity of *PrecRep-2* can be improved for example by memorizing how much time is actually available for backward shifts (to prevent the stretch stage).

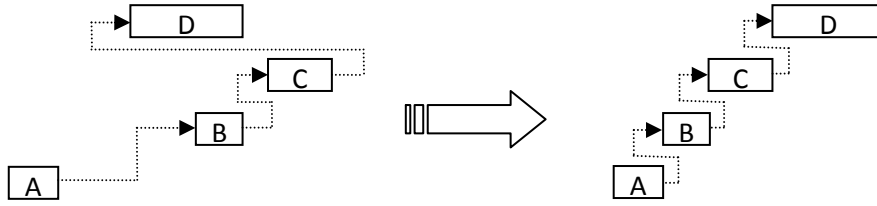


Fig. 4. Algorithm *PrecRep-2* exploits better available time on left of D.

4.3 Resource capacity repair technique

The final stage of the proposed repair algorithm consists of repairing resource conflicts. Recall that activities require for their processing unary resources; it is possible that an activity requires more than one resource (for example machine, tool, and worker). There is a resource conflict if two (or more) activities require the same resource at the same time.

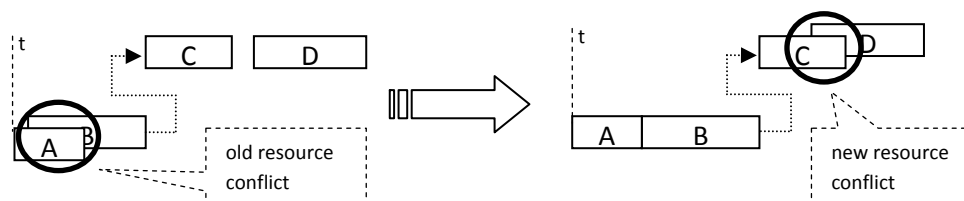
From the previous stage we have a schedule that does not violate precedence constraints so it is either feasible or some resource constraints are violated. We now present a technique that repairs resource conflicts while keeping the precedence constraints satisfied. This technique resolves the conflict by shifting one of the activities forward in time. The algorithm *ResRep* iteratively repairs resource conflicts and each time a new precedence conflict is introduced then all precedence conflicts are repaired before continuing to the next resource conflict. By sweeping the schedule from past to future we remove all violated constraints (recall that there are no deadlines so any activity can be shifted forward).

```

algorithm ResRep
1  while any constraint is violated do
2    if precedence is violated then
3      select violated precedence ( $A \rightarrow B$ ) with smallest  $s_A$ 
3    else
4      let A,B be activities violating resource constraint (2)
5      such that  $s_A \leq s_B$  and  $s_A$  is smallest among such pairs
6    end if
7     $s_B \leftarrow s_A + d_A$ 
8  end while
end ResRep
    
```

Proposition 3: Algorithm *ResRep* is sound and complete (produces a feasible schedule).

Proof: If the algorithm stops then there are no violated constraints in the schedule (because of the condition in the while loop) so the final schedule is feasible and the algorithm is sound. To prove completeness, it is necessary to show that the algorithm stops after a finite number of iterations (recall that a feasible schedule always exists). We prove the proposition by showing that the number of possible conflicts decreases as we sweep the schedule from past to future. At the beginning, no precedence constraints were violated so some resource constraint must be violated. Let t be the smallest time such that for some activity A with $s_A = t$ there is a violated resource constraint (2) between A and B . The constraint is repaired by shifting B forward to start after activity A (line 7 of the algorithm), which may violate some precedence constraints ($B \rightarrow C$). These violated precedence constraints will be repaired in next iterations of the algorithm (repairing precedence constraints is preferred to repairing resource conflicts) by shifting C forward etc. It may happen that during these repairs some new resource conflicts are introduced. However, all these new resource conflicts “start” after time t ; formally, for any new resource conflict between activities C and D , $t < s_C$ and $t < s_D$ hold. The reason is that the conflicting activity, say C , that was shifted forward by the precedence repair steps now starts after $s_A + d_A$. The other activity D must start after t as well because otherwise there was already a resource conflict between C and D .



In summary, after repairing the resource conflict between A and B starting at time t and “propagating” it to all precedence constraints, no other resource conflict starting at or before t was added. As there is a limited number of resource conflicts, after a finite number of iterations, all resource conflicts starting at t are repaired and one of the former conflicting activities still starts at t . Now we can move to the next resource conflict which starts at time $t' > t$. Because the number of activities starting at or after t' is strictly smaller than the number of activities starting at or after t , the maximal number of possible conflicts is also smaller. This upper bound can be simply the number of all pairs of activities starting at or after t . Hence by shifting forward in time, the upper bound on the number of possible conflicts decreases and therefore the algorithm repairs all conflicts after a finite number of iterations. ■

5 Conclusions

The paper proposes a novel local repair technique for correcting violated precedence and resource constraints in RCPSP-like scheduling problems. By doing local repair steps, the schedule changes only locally in the area of the flaw, which is the main advantage over re-scheduling from scratch. Moreover, the proposed techniques are fully automated and problem independent so it is not necessary to describe specific repair rules for the problem as for example in modified Affected Operations Rescheduling [9]. To repair violated precedence constraints, we can shift activities to left (past) which keeps the schedule more compact in comparison with Right shift rescheduling [4]. Last but not least, the proposed method can repair any number of precedence and resources conflicts while traditional schedule repair algorithms work with a single disruption. Hence our method is appropriate for repairing schedules where several disruptions are scattered in time and on resources, which is typical for manually modified/constructed schedules.

We implemented the proposed techniques within an interactive Gantt viewer (Figure 1). Thought that exist many interactive Gantt viewers we are not aware about anyone providing automated schedule repair. Hence the main focus of the paper is on the formal description of the repair techniques and on theoretical justification of their soundness and completeness.

All presented techniques exploit the feature of the problem that there are no deadlines so it is always possible to shift activity forward in time. Nevertheless, when repairing the precedence constraints we tried to exploit also free time before the activity, that is, to shift activities backward. The main motivation was to keep the schedule as compact as possible (not to increase makespan a lot). We proposed two alternative repair techniques with the tradeoff between the compactness of the schedule and the speed of the technique. The technique producing more compact schedules can probably be speeded-up by using additional information during computation. This is a topic of future work. The current technique for repairing resource conflicts shifts activities only forward so another open question is whether the ideas from precedence repairs (backward shifts) can be used there.

Acknowledgements The research is supported by the Czech Science Foundation under the contract no. 201/07/0205.

References

1. Abumaizar RJ, Svestka JA, Rescheduling job shops under random disruptions. *International Journal of Production Research* 35(7):2065–2082 (1997)
2. Barták R, Müller T, Rudová H, Minimal Perturbation Problem – A Formal View. *Neural Network World* 13(5): 501–511 (2003)
3. Błazewicz J, Lenstra JK, and Rinnooy Kan AHG. Scheduling projects to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24 (1983)
4. Brandimarte P, Rigodanza M, Roero L, Conceptual modeling of an object oriented scheduling architecture based on the shifting bottleneck procedure. *IIE Transactions* 32(10):921–929 (2000)
5. El Sakkout H, Richards T, Wallace M, Minimal Perturbation in dynamic scheduling. *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI98)*. John Wiley & Sons (1998)
6. Kocjan W, Dynamic Scheduling – State of the Art Report. SICS Technical Report T2002:28. SICS (2002)
7. Müller T, Barták R, Rudová H, Iterative Forward Search Algorithm: Combining Local Search with Maintaining Arc Consistency and a Conflict-Based Statistics. *LSCS'04 - International Workshop on Local Search Techniques in Constraint Satisfaction* (2004)
8. Oddi A, Policella N, Cesta A, Smith SF, Boosting the Performance of Iterative Flattening Search. *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, LNCS 4733, pp. 447–458, Springer Verlag (2007)
9. Subramaniam V, Raheja AS, mAOR: A heuristic-based reactive repair mechanism for job shop schedules. *The International Journal of Advanced Manufacturing Technology* 22: 669–680 (2003)

Preemptive scheduling with precedences and alternative resources

Vladimír Duša · Roman Barták

Abstract This paper presents a search algorithm solving an elastic scheduling problem with alternative resources, where a feasible schedule with minimal costs on resources is demanded. The proposed technique is based on detecting "problematic" precedence constraints defined between activities processed on different resources. Assignment of alternative resources is realized as an extension of the search algorithm using overload checking. Preliminary experimental results show, that the proposed technique works well for problems with a smaller number of precedence constraints.

1 Introduction

Nowadays it is more and more important to save as many resources as possible primarily in larger companies, where production costs are measured in hundreds thousands of dollars per month so in such companies each percent of savings can be very significant. The assumed resources can be people in offices, machines in production, runways at the airport etc.

Scheduling of (usually) human resources will be discussed in this paper. These resources can be people working for example in construction or production processes or office workers, whose work schedule depends on various projects. We do not need to schedule their work in tiny details, but we only need to know if all resources have enough time to process all given activities and when these activities start and when they are finished. Thus we can schedule their work as if they could process the activities simultaneously even if they do not work in such a way in reality. People can usually interrupt their work and do something else in the meanwhile. For example an office worker can process more projects simultaneously – he can begin with one activity, then he must wait (for example for an answer from a customer) and in the meanwhile he can process some other activity. The project manager usually knows what type of worker

Vladimír Duša
Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
E-mail: dusa@ktiml.mff.cuni.cz

Roman Barták
Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
E-mail: bartak@ktiml.mff.cuni.cz

he needs to process given activity, but it is not interesting who the person exactly is in the planning phase. For example the project manager knows, that for a building of a wall he needs a bricklayer for five days. At this moment it is not important, in fact, who the person is. Also we know price of each resource work unit. For example one bricklayer can cost 5 USD per hour but another can cost 10 USD per hour. Our goal is to find a feasible schedule according to given constraints with the lowest possible costs on assigned resources.

This paper describes a new approach for solving preemptive scheduling problems with alternative resources based on detecting "problematic" precedences between activities. There has been intensive research in non-preemptive scheduling, but preemptive scheduling problems have received limited attention. Some results have been published in [1], [2] and [3], where an approach using constraint programming for preemptive, non-preemptive and mixed case is described. Another search algorithm solving preemptive scheduling problem has been introduced in [5], where heuristic search has been used. In [6] an approach minimizing changes in the schedule after adding a new activity is introduced. The problem of alternative resources has been studied in [1], where an "optional activity" for each alternative resource is created, or also in [4], where the problem of alternative resources in non-preemptive scheduling is presented. None of these papers deal with preemptive scheduling with alternative resources and objective function minimizing total cost on used resources together which is the main topic of this paper.

A formal definition of the problem will be given in the next section. Then the problem will be transformed to a problem of preemptive scheduling and it will be shown, that it belongs to the class of NP-Complete problems. After that an algorithm solving the preemptive problem will be introduced – firstly for the problem with only one resource, then for the problem with more resources (where each activity has assigned exactly one resource) and finally for the problem with alternative resources. Some preliminary experimental results will be presented before conclusions.

2 Problem definition

2.1 Terms

As it has been mentioned, scheduling of a (usually) human resources will be assumed. Activities to be processed will be *elastic*, which means that there is no need of constant workload on such activity, but the workload of the resource during processing the activity can vary in time between zero and maximum capacity of the resource. Resources processing these activities will be *cumulative*, which means, that one resource can process more activities at a time. Naturally the resources can not be overloaded. We have given a set of *alternative resources* for each activity, from which exactly one resource will process the activity. Each activity needs a given amount of *energy* of this resource to be processed. Later we will use the term *preemptive scheduling*, which means that all resources have maximum capacity equal 1 (the resource can process at most one activity at a moment) and activities can be interrupted during their processing.

2.2 Problem given as elastic scheduling

Our scheduling problem is defined as follows: there is a finite set of tasks A to be processed on a finite set of cumulative resources R . Capacity of resource $k \in R$ is denoted as $cap(k) \in \mathbb{Z}^+$, $cost(k) \in \mathbb{Z}$ denotes the price of one energy unit of resource k . We assume discrete time given by time points $t \in \mathbb{Z}^+$. For each activity $i \in A$, r_i denotes the earliest possible start time (release time) and d_i the latest possible end time (deadline). Let e_i be the energy needed to process activity i and $e_i(t)$ denotes the current load of the resource assigned to activity i between time points t and $t+1$ (for example see figure 1 where $e_i(r_i) = 0, e_i(r_i + 1) = 3, e_i(r_i + 2) = 4, \dots$). Let $M_i \subseteq R$ be a set of *alternative resources* of activity i . Between activities $i, j \in A$ a precedence constraint $i \rightarrow j$ can be defined, meaning $end(i) \leq start(j)$, where $start(i) = \min\{t \mid e_i(t) > 0\}$, respectively $end(i) = \max\{t \mid e_i(t) > 0\} + 1$. Activity i is called a *predecessor* of activity j and analogically j is called a *successor* of activity i . We suppose that precedences do not create a circle such as $i \rightarrow j \rightarrow \dots \rightarrow i$. Each activity $i \in A$ must be processed by exactly one resource from its set of alternative resources. Let this resource be denoted as $R(i)$. The goal is to find exactly one resource $R(i) \in M_i$ and $e_i(t)$ for each $i \in A$ and for each time point t such that:

1. No precedence constraint is violated.
2. Each activity i is processed in its time window that is $\forall i \in A, \forall t : t < r_i \vee t \geq d_i \Rightarrow e_i(t) = 0$.
3. No resource is overloaded:

$$\forall t, \forall k \in R : \sum_{i \in A : R(i)=k} e_i(t) \leq cap(k)$$

4. There is enough energy allocated to each activity:

$$\forall i \in A : \sum_t e_i(t) = e_i$$

5. Total cost of used resources $\sum_{i \in A} e_i \cdot cost(R(i))$ is minimal.

2.3 Transformation of elastic scheduling to preemptive scheduling

A problem of elastic scheduling has been defined in the previous section. Let us assume that we have only one resource in the problem (that is $|R| = 1$). The problem of elastic scheduling on one resource can be easily transformed to preemptive scheduling problem on one resource. Let us note that it is not common to use the term *energy* e_i in the preemptive scheduling and instead of it the term *processing time* p_i is used. Also the variable $e_i(t)$ collapses to a boolean variable $X_i(t)$, that is equal 1 if the activity is being processed between the time points t and $t+1$ (a so called timetable variable). Otherwise it holds $X_i(t) = 0$. Now we will describe the transformation between elastic and preemptive scheduling on one resource (according to [1]):

Transformation 1 For an arbitrary problem of elastic scheduling P , let $F(P)$ be a problem of preemptive scheduling on one resource k with maximum capacity $cap(k)$ defined on n activities $1', \dots, n'$, where $\forall i \in \{1, \dots, n\} r'_i = cap(k)r_i, d'_i = cap(k)d_i, p'_i = e_i$, where p'_i denotes processing time of preemptive activity i' .

Theorem 1 [1] *For an arbitrary problem of elastic scheduling P there exists a feasible schedule if and only if there exists a feasible schedule for $F(P)$.*

If there are more resources in the schedule, as in our case, we must modify the above transformation. Let us assume now, that each activity in our problem has assigned exactly one resource processing the activity (that is $\forall i \in A : |M_i| = 1$). Then we can transform problem P to $F(P)$ by multiplying the release time and the deadline of each activity by $maxcap = \max_{k \in R} cap(k)$. All resources $k \in R$ with capacity $cap(k) < maxcap$ must be occupied by some "dummy" activities at specified locations, which ensure that the resource will not be overloaded. On figures 1 and 2 we can see the transformation from elastic to preemptive schedule. There must be created a "dummy" activity for each succession time points of the preemptive schedule for each resource. Such "dummy" activity $d \in \{1, 2, \dots\}$ for resource k will have $M_d = \{k\}$, $r_d = (d - 1) \cdot maxcap + cap(k)$, $d_d = maxcap \cdot d$ and $p_d = d_d - r_d$ (these activities are shown in the figure 2 as crossed parts of the schedule). All these "dummy" activities will be removed as soon the schedule is created. Thus we can look on the problem of elastic scheduling on more resources as it would be a problem of preemptive scheduling on more resources. We do this transformation, because it is easier to work with preemptive scheduling than with elastic scheduling.

Fig. 1 Elastic schedule of a resource k

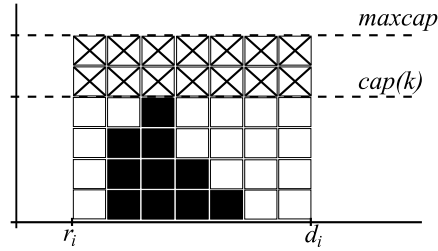
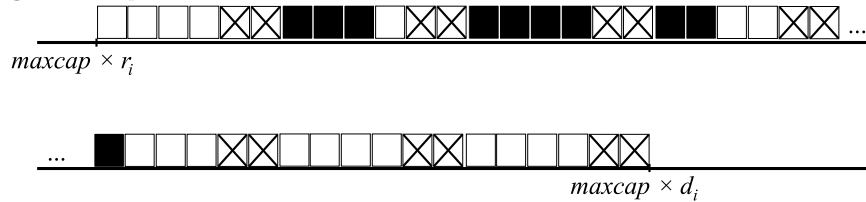


Fig. 2 Preemptive schedule of a resource k transformed from elastic schedule



2.4 Problem hardness

The problem of preemptive scheduling introduced in the previous section is a problem belonging to the class of NP-Complete problems. The problem marked as [SS12] defined in [7] is proved to be a member of NP-Complete class:

[Preemptive scheduling problem SS12] Let A be a set of activities with precedence constraints, $m \in \mathbb{Z}^+$ be a set of resources, $p_i \in \mathbb{Z}^+$ be processing time of each activity $i \in A$ and D be a common deadline for all activities. Question: Is there a feasible preemptive schedule on m resources for A satisfying all precedence constraints and deadline D ?

There exists no polynomial algorithm for our scheduling problem defined in 2.3 unless $P = NP$, which is obvious when we consider an instance of our problem with $r_i = 0$ and $d_i = D$ for all $i \in A$.

3 Solving approaches

We will introduce an algorithm solving a special case of preemptive problem containing only one resource in the schedule ($|R| = 1$) in the first part of this section. Then we will introduce an incomplete¹ algorithm for another special case with more resources but no alternative resources ($|R| \geq 1$ and $\forall i \in A |M_i| = 1$). Finally, we will propose a branch-and-bound algorithm for solving the problem with more resources and alternative resources for activities.

3.1 Problem with only one resource

For the problem of preemptive scheduling with one resource and no precedence constraints (defined as $1|r_i; d_i; prmp|$ in $\alpha|\beta|\gamma$ notation), we can use the well known *earliest due date* algorithm [8] which creates a so called *Jacksons preemptive schedule (JPS)* [1]. This algorithm has one simple rule also known as earliest due date: For each time point t , whenever an activity becomes available, schedule not yet finished activity i with lowest d_i (activity i is available in time points $t \geq r_i$). As soon as the schedule contains precedence constraints (but still only one resource), we can use the following modification of the earliest due date algorithm, which is also a complete algorithm for the problem with one resource and precedence constraints ($1|r_i; d_i; prmp; prec|$):

3.2 Problem with no alternative resources

Until now, we considered only one resource in the problem. If there are more resources in the problem ($M_i|r_i; d_i; prec; prmp|$ with $|M_i| = 1$), we can modify algorithm 1 as follows:

If we try to create a schedule using algorithm 2, a feasible schedule may not be found even though it exists. Figure 3 shows an example of four activities where $p_1 = 2$, $p_2 = 4$, $p_3 = 4$, $p_4 = 3$ are being processed on two resources $k_1, k_2 \in R$. On the left side of figure 3 an infeasible schedule created by algorithm 2 is shown. However as we can see on the right side of figure 3, a feasible schedule exists.

The precedence constraints defined between activities being processed on different resources make the problem hard. If we would have an instance of the problem, containing no precedences between activities being processed on different resources, we could create the schedule separately for each set of activities processed on the same

¹ We say, that an algorithm is complete if it guarantees finding a feasible solution if some exists. Otherwise we say, that the algorithm is incomplete.

Algorithm 1: Jacksons preemptive schedule (JPS)

input : Scheduling problem instance P
output: Feasible schedule if it exists, otherwise null

while *there exists precedence constraint* $i \rightarrow j$ *such that* $d_i > d_j - p_j$ **do**
 $d_i \leftarrow (d_j - p_j)$
end

for $t = \min_{i \in A}(r_i) \dots \max_{i \in A}(d_i - 1)$ **do**
 $i \leftarrow$ any not yet finished available activity $i \in A$ with smallest d_i
 if $t = d_i$ **then**
 return fail
 end
 if i *exists* **then**
 schedule activity i in time $\langle t, t + 1 \rangle$
 end
end

return schedule

Activity i is available in time point t , when $t \geq r_i$ and all predecessors are finished.
Activity i is finished if it has been scheduled to p_i time points

Algorithm 2: JPS with more resources

input : Scheduling problem instance P
output: Feasible schedule if it exists, otherwise null

while *there exists precedence constraint* $i \rightarrow j$ *such that* $d_i > d_j - p_j$ **do**
 $d_i \leftarrow (d_j - p_j)$
end

for $t = \min_{i \in A}(r_i) \dots \max_{i \in A}(d_i - 1)$ **do**
 foreach $k \in R$ **do**
 $i \leftarrow$ any not yet finished available activity $i \in A(k)$ with smallest d_i
 if $t = d_i$ **then**
 return fail
 end
 if i *exists* **then**
 schedule activity i in time $\langle t, t + 1 \rangle$
 end
 end
end

return schedule

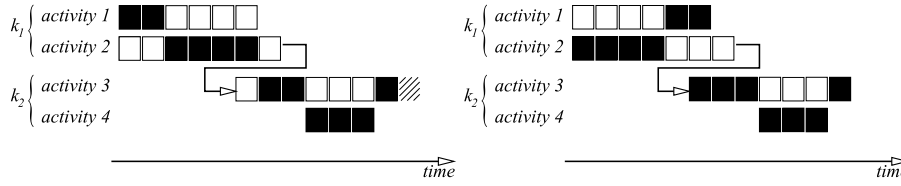
Activity i is available in time point t , when $t \geq r_i$ and all predecessors are finished.
Activity i is finished if it has been scheduled to p_i time points

resource. Such algorithm would be complete, because no activity processed on one resource affects any other activity processed on another resource. Thus we shall focus on the precedence constraints defined between activities being processed on different resources.

Definition 1 Let A be a set of activities and $P = \{i \rightarrow j | i, j \in A\}$ be a **set of all precedence constraints**. Set $\tilde{P} \subseteq P$ defined as $\tilde{P} = \{i \rightarrow j \in P | R(i) \neq R(j)\}$ is called a **set of inter-resource precedence constraints**. Precedence constraint $i \rightarrow j \notin \tilde{P}$ will be called **intra-resource precedence constraint**.

Theorem 2 Let $\tilde{P} = \emptyset$, then algorithm 2 is complete for the scheduling problem with more resources and precedence constraints $(M_i | r_i; d_i; \text{prec}; \text{prmp})$ with $|M_i| = 1$.

Fig. 3 Incompleteness of JPS with more resources



Proof Thanks to the fact, that $\tilde{P} = \emptyset$, algorithm 2 works as if the schedule would be obtained for each resource separately by algorithm 1, because none of the schedules of any resource affects schedule of another resource. Algorithm 1 is complete, thus algorithm 2 is complete as well. $\square$

For the search algorithm, that will be introduced later, we need as few as possible precedence constraints in $\tilde{P}$. Now we will show, that not all of the inter-resource precedence constraints from $\tilde{P}$ must be "problematic" and we will construct a smaller set of precedence constraints P^{crit} .

Definition 2 Let A be a set of activities and R be a set of resources. Let $k \in R$ be a resource. Then $A(k) = \{i \mid i \in A, k \in M_i\}$ denotes **set of activities being possibly processed on resource k** .

Definition 3 Let A be a set of activities with precedence constraints and R be a set of resources. Let $A^-(i) = \{l \in A(R(i)) \setminus \{i\} \mid d_l > r_i + p_i \wedge d_l \leq d_i\}$. Let $i \rightarrow j \in \tilde{P}$ be an inter-resource precedence constraint. Algorithm 2 has created a partial schedule, where activity $f \in A$ has failed² to be scheduled then we call $i \rightarrow j$ **critical precedence constraint according to f** in the partial schedule, if the following conditions are fulfilled:

1. $d_j \leq d_f$
2. $A^-(i) \neq \emptyset$
3. $\exists$ time point $t \geq \max\{r_i + p_i, r_j\}$ such that $X_i(t) = 1$
4. For some t satisfying point 3 one of the following conditions holds
 - (a) $\exists m \in A$ such that $R(m) = R(j)$ and $d_m \geq d_j$ and $X_m(t) = 1$
 - (b) $\forall m \in A(R(j))$ holds $X_m(t) = 0$.

$\square$

Let us explain definition 3 in more detail: $A^-(i)$ contains all activities different from i being processed on the same resource as i , such that their deadlines lie between the earliest end time of i and deadline of i . Such activities can be postponed to allow earlier finish of i . The point 2 of definition 3 says that predecessor i has not been processed in the partial schedule with the highest priority (there is another activity with earlier deadline overlapping i and being processed on the same resource). Moreover activity i is being processed in time point t , when successor j could already be processed, which the point 3 says. It means, that there is a chance to finish activity i earlier, if it would have higher priority in the algorithm 2, that created the partial schedule. This would allow earlier start of successor j . However successor j could start earlier only, if resource

² activity $i \in A$ fails if there is not enough time to process i in its time window $\langle r_i, d_i \rangle$

$R(j)$ is processing another activity m with lower priority (that is $d_m \geq d_j$) in time point t (point 4a) or if the resource does not process any activity in time point t (point 4b).

Briefly speaking the precedence constraint $i \rightarrow j$ is critical, if activity i is being processed in the time point, when activity j would be processed if there was no precedence constraint between i and j and there is a possibility to finish activity i earlier.

Notation 1 Let A be a set of activities with precedence constraints and R be a set of resources. Let algorithm 2 has created a partial schedule, where an activity $f \in A$ has failed to be scheduled. We will denote $P^{crit}(f)$ as a **set of all critical precedence constraints according to activity f** .

Theorem 3 Let there be a scheduling problem with more resources and precedence constraints $(M_i|r_i; d_i; prec; prmp|$ with $|M_i| = 1)$. Let algorithm 2 has created a partial schedule of this scheduling problem, where an activity $f \in A$ has failed to be scheduled. If it holds $P^{crit}(f) = \emptyset$, then there does not exist a feasible schedule of the given scheduling problem.

Proof As we have seen in figure 3, the earlier finish of activity 2 allowed earlier start of activity 3, which led to a feasible schedule. Firstly we will show, that the predecessors i in non-critical inter-resource precedence constraints $i \rightarrow j$ can not be finished earlier in the partial schedule created by algorithm 2.

Let us consider inter-resource precedence constraint $i \rightarrow j \in \tilde{P}$, such that $d_j \leq d_f$. Since $P^{crit}(f) = \emptyset$, one of the following conditions holds for $i \rightarrow j$:

- C1: $A^-(i) \neq \emptyset$: If condition 3 of definition 3 is not fulfilled then i does not occupy any time that can be used by j thus i does not delay j . Otherwise, if neither 4a nor 4b in definition 3 is fulfilled then the delay of j is caused by activities on resource $R(j)$ with higher priority than j , but not by activity i . Hence even if activity i finishes earlier, it would not allow earlier start of activity j .
- C2: $A^-(i) = \emptyset$: Since $A^-(i) = \emptyset$, then it is not possible to finish activity i earlier, because activity i has been processed with highest priority in the partial schedule, thus it was finished as soon as possible. Hence $end(i) \leq start(j)$, there was also no way to start activity j earlier.

Let us assume that failed activity f was processed on resource k . Now we will check if activity f could be finished earlier in the partial schedule:

1. Either $d_f - r_f < p_f$ then there is not enough time to finish activity f in its time window.
2. Or $d_f - r_f \geq p_f$ then activity f was delayed and failed, because there is at least one time point $t \in \langle r_f, d_f \rangle$ such that $X_f(t) = 0$. Let us consider these time points t in descending order. Question is: why did not algorithm 2 schedule activity f in time point t ?
 - (a) Either there is no activity processed in time point t on resource k . Then there must exist inter-resource precedence constraint $i \rightarrow f$ where activity i is not yet finished in time point t . According to condition C1 and C2 activity i could not be finished earlier thus activity f can not be processed in time point t .
 - (b) Or in time point t resource k processes an activity $l \neq f$ such that
 - either $d_l > d_f$: according to the rules of algorithm 2 there is in time point t non finished activity i with inter-resource precedence constraint $i \rightarrow f$

(otherwise algorithm 2 would schedule activity f in time point t). In this case we can use same idea proposed in point 2a. Let us note, that according to the preprocessing of deadlines in algorithm 2 there can not exist intra-resource precedence constraint $i \rightarrow f$ such that i is not finished in time point t , while activity l is being processed in time point t . If an intra-resource precedence constraint $i \rightarrow f$ existed it would hold $d_i < d_f$. Since $d_l > d_f$, it holds $d_i < d_l$ which implies, that activity i would be scheduled in time point t instead of activity l , however it did not happen.

- or $d_l = d_f$: Since activity f has failed it holds $d_f < \text{end}(f)$ in the partial schedule. If activity f was scheduled in time point t instead of activity l , then this missing time point of activity l has to be scheduled in time point $\text{end}(f) - 1$ or later in order to finish activity f earlier. Since $\text{end}(f) - 1 \geq d_f = d_l$, then activity l would fail. Thus activity f can not be scheduled in time point t .
- or $d_l < d_f$: according to the rules of JPS activity l must be processed with higher priority and activity f could not be processed in time point t . Now there is a question if it was possible to finish activity l earlier in order to schedule activity f in time point t . Answer to this question gives us same consideration for l that we made for f now. So we can iteratively go through activities processed on resource k until we meet any inter-resource precedence constraint or begin of the schedule in time point 0 proposing that no change in the schedule sustaining feasibility can cause earlier finish of activity f .

□

Now a search algorithm will be introduced, that finds a feasible schedule if such a schedule exists (see algorithm 3). This algorithm uses algorithm 2, that tries to create the schedule from the beginning to the end. In the moment, when algorithm 2 finds some activity f that fails, algorithm 3 looks into the set of critical precedence constraints and tries to change time allocation of some activities predecesing f , which can positively affect the time allocation of f . There are several ways how to do it so the search algorithm needs to explore them.

Let us clarify algorithm 3 in more detail. First of all we try to create the schedule using algorithm 2. Then if no feasible schedule has been found, algorithm 2 stopped at first activity f , that failed. According to this activity we analyze the set of critical precedence constraints, that could affect activity f . All such precedence constraints are members of set $P^{crit}(f)$. If this set is empty, there is no feasible schedule and we must backtrack. If $P^{crit}(f)$ is a nonempty set, then we will try to finish earlier the activities, that are affecting the failed activity f . We will do it in such a way, that we will successively decrease the deadline of all activities i such that $i \rightarrow j \in P^{crit}(f)$. Decreasing of the deadlines causes branching of the algorithm and is performed as follows: Let $i \rightarrow j \in P^{crit}(f)$ and let the set $A^-(i)$ contains m activities, then we will try to change the deadline of activity i in m ways: for each activity $q \in A^-(i)$ we assign $d_i := d_q - 1$. In such a way we must try all combinations of such changes of the deadline of all activities i , where $i \rightarrow j \in P^{crit}(f)$. Change of deadline of only several activities is also considered as a combination. Decreasing of deadline involves new ordering of the activities in algorithm 2, that may allow earlier finish of some predecessor of the failed activity f .

Algorithm 3: Search algorithm

```

input : Scheduling problem instance  $P$ 
output: Feasible schedule if it exists, otherwise null

 $schedule \leftarrow \text{Algorithm2}(P)$ 
if  $schedule$  is feasible then
    | return  $schedule$ 
else
    |  $f \leftarrow$  failed activity in  $schedule$ 
    | if  $P^{crit}(f) \neq \emptyset$  then
    |     | foreach problem modification  $P'$  of problem  $P$  according to  $P^{crit}(f)$  do
    |     |     |  $schedule \leftarrow \text{Algorithm3}(P')$ 
    |     |     | if  $schedule \neq \text{null}$  then
    |     |     |     | return  $schedule$ 
    |     |     | end
    |     | end
    |     | return null (backtrack)
    | else
    |     | According to theorem 3 there is no feasible schedule for the problem  $P$ 
    |     | return null (backtrack)
    | end
end

```

Since the branching factor can be large (depending on the number of critical precedence constraints), the search space can be huge: $O(2^{|\bar{P}| \cdot (n-1)})$, where $n = |A|$ (that is upper bound assuming that all inter-resource precedence constraints $i \rightarrow j$ are critical, while $A^-(i)$ contains all other activities than i (that is $|A^-(i)| = n - 1$). Then we have to check $n - 1$ changes of deadline of predecessor i). Let us note that not all critical precedence constraints are needed to be checked. Let us assume, that the search algorithm is in the depth r of the search tree, and that activity f has failed. Then we do not need to check all critical precedence constraints from the set $P^{crit}(f)$, but only the "new" critical precedence constraints, that are not members of the set of critical precedences in upper levels of the search tree. These precedence constraints will be used for branching later (or have been used already) in the upper levels of the search tree.

3.3 Solving the optimization problem with alternative resources

3.3.1 Applying branch-and-bound technique

In the previous section we have presented search algorithm solving the special case of the problem where $|M_i| = 1 \ \forall i \in A$ (each activity has assigned exactly one resource that will process the activity). Our goal is to solve the problem, where for each activity i a resource from the set of alternative resources having $|M_i| \geq 1$ must be chosen.

Let $X = \{R(1), \dots, R(n)\}$ be a set of variables, where $R(i)$ denotes the resource, that processes activity i and $n = |A|$. Let $M = \{M_1, \dots, M_n\}$ be a set of sets of alternative resources. Let the resources in each set of alternative resources be ordered non descending according to their costs.

We denote our scheduling problem with alternative resources as a tuple $SP = (A, P, R, M)$, where A is a set of activities, P is a set of precedence constraints between the activities, R is a set of resources and M is a set of all sets of alternative resources

(defined above). When solving problem SP , we can divide it into two subproblems $SP' = (A, P, R, M')$ and $SP'' = (A, P, R, M'')$, where $M' = \{M_1, \dots, M'_i, \dots, M_n\}$ and $M'' = \{M_1, \dots, M''_i, \dots, M_n\}$ and it holds³ $M_i = M'_i \dot{\cup} M''_i$ and solve each subproblem separately. Now we can solve both subproblems separately. In particular, whenever activity i having $|M_i| > 1$ is scheduled by algorithm 3, we choose the cheapest resource $k \in M_i$ and we divide the problem into two subproblems where $M'_i = \{k\}$ and $M''_i = M_i \setminus \{k\}$. Then we solve these subproblems separately with algorithm 3. We start with the cheapest resource k for each activity as a simple greedy heuristic for minimizing total cost.

A bounding technique pruning the search space will be introduced now. A lower bound $LB(SP)$ defined below will be checked during the search

$$LB(SP) = \sum_{i \in A} p_i \cdot \min \{cost(k) \mid k \in M_i\}$$

This lower bound $LB(SP)$ is calculated as if each activity in the problem SP would be processed on the cheapest resource from its set of alternative resources.

We remember the cost of the best solution found so far. It will be denoted as LB^* and at the very beginning we set $LB^* = \infty$. The lower bound LB will be calculated for both subproblems during each branching. If it holds $LB(SP') \geq LB^*$, (or $LB(SP'') \geq LB^*$), we do not need to solve the subproblem SP' , (or SP''), because it does not contain a better solution than the best solution found so far.

3.3.2 Resource consistency check

In the previous section we have introduced an extension of algorithm 3, that solves the problem with alternative resources. This extension expands the branching factor of algorithm 3 because for each resource in each set of alternative resources a new branch is created. Size of the search space is now $O(2^{|\tilde{P}| \cdot (n-1)} \cdot \prod_{i \in A} |M_i|)$. Whenever a set of alternative resources M_i contains exactly one resource k , we can check if it is possible to schedule parallel activities on the same resource k .

In [9] an overload checking for cumulative resources based on task intervals has been introduced. We denote $p(M) = \sum_{i \in P} p_i$, $r_M = \min_{i \in M} r_i$ and $d_M = \max_{i \in M} d_i$ for $M \subseteq A$. We define a set $P_{ij} = \{l \in A \mid r_l \geq \min\{r_i, r_j\}, d_l \leq \max\{d_i, d_j\}\}$ for each $i, j \in A$.

Resource consistency check is a technique finding P_{ij} over $i, j \in A$ such that $r_{P_{ij}} + p(P_{ij})$ is maximal. If it holds $p(P_{ij}) > d_{P_{ij}} - r_{P_{ij}}$ for such P_{ij} , then no feasible schedule exists. For more detailed description see [9].

Now we can use the above ideas to easily perform overload checking in the algorithm 3 as follows: Whenever a resource is removed from set of alternative resources M_j during the search procedure described in the previous section and only last resource k remains in M_j , we will check if the resource k is not overloaded. We perform this check by finding set P_{ij} such that $r_{P_{ij}} + p(P_{ij})$ is maximal. Let us note, that the set P_{ij} is constructed only over activities $l \in A$ having $M_l = \{k\}$ in the current node of the search tree mentioned in section 3.3.1. If it holds $p(P_{ij}) > d_{P_{ij}} - r_{P_{ij}}$ for such P_{ij} then there does not exist a feasible schedule where resource k processes activity i and the current branch can be pruned.

³ $U \dot{\cup} V$ denotes disjunctive union (that is $U \cap V = \emptyset$)

4 Preliminary experimental results

Performance of the algorithm described in the previous sections has been tested on randomly generated data on AMD Duron 850 MHz. The code has been written in C++. The data has been generated as follows⁴:

- Number of activities has been given by constant (50)
- Number of all resources has been given by constant (10)
- Capacity of each resource has been given randomly (4 ± 2)
- Number of alternative resources for each activity has been given randomly (7 ± 2)
- Processing time of each activity has been given randomly (14 ± 2)
- Size of the time window of each activity has been given randomly (10 ± 4)

Since the search algorithm is sensitive to the number of precedence constraints, all parameters but number of precedence constraints has been fixed as described above. There has been generated 4×40 instances of the problem (40 instances with 100 precedence constraints, 40 instances with 200 precedence constraints, etc). Results are shown in table 1.

Table 1 Results of measurement

Number of precedences	Number of solutions Optimum / Stopped ⁵	Average time ⁶	Average distance from lower bound ⁷
100	35/5	6:40	3 %
200	27/13	7:59	4 %
300	16/24	9:00	7 %
400	8/32	9:24	4 %

According to the assumption, results indicates that the search algorithm is sensitive on the number of precedence constraints. The smaller number of precedence constraints in the instance was the more the algorithm was "successful". More precedence constraints increases probability of being critical, which expands the search space.

5 Conclusion

A search algorithm solving elastic respectively preemptive scheduling problem with alternative resources and minimizing resource cost has been introduced. This search algorithm based on a detection of "problematic" precedence constraints tries to finish predecessors of a failed activity earlier, which can help the failed activity to be correctly scheduled. Preliminary experiments have shown that the proposed algorithm is sensitive to the number of precedence constraints, as we expected. The open question is whether

⁴ Numbers of activities, resources, capacities and sizes of alternative resources sets were taken from average real life projects

⁵ Optimum means that the algorithm have finished before 10 minutes time limit.

⁶ Average time has been calculated only from calculations reached optimum, that were longer than 1 second.

⁷ Average distance from the lower bound has been calculated as $\frac{Sol - Min}{Max - Min}$, where *Sol* denotes the costs of the best solution found, *Min* equals to the *LB* (defined in 3.3.1) of the given instance of the problem. *Max* was defined analogically to the *Min*.

it is possible to further reduce the number of "problematic" precedence constraints by their more careful identification.

The branching factor in algorithm 3 depends also on the number of all possible changes of the deadlines of activities, that can affect the failed activity. Future research may focus on a heuristic function that predicts which changes of deadlines have higher impact on finding a feasible schedule.

Acknowledgements The research is supported by the Czech Science Foundation under the contract no. 201/07/0205.

References

1. Baptiste P., Le Pape C., Nuijten W., Constraint based scheduling: applying constraint programming to scheduling problems, Kluwer Academic Publishers, Boston, MA, (2001)
2. Baptiste P., Le Pape C., Resource Constraints for Preemptive Job-shop Scheduling, Constraints: An international Journal, 3, 263-287 (1998)
3. Le Pape C., Baptiste P., Heuristic Control of a Constraint-Based Algorithm for the Preemptive Job-Shop Scheduling Problem, Journal of Heuristics, Vol. 5, 305-325, (1999)
4. Wolf A., Schlenker H., Realising the Alternative Constraint, LNCS, 3392/2005, 185-199 (2005)
5. Zhao W., Ramamritham K., Stankovic J. A., Preemptive Scheduling Under Time and Resource Constraints, IEEE Transaction on Computers, Vol. C-36, No. 8, (1987)
6. Zhou Q., Smith S. F., A Priority-Based Preemption Algorithm for Incremental Scheduling with Cumulative Resources, Robotics Institute, Carnegie Mellon University (2003)
7. Garey M. R., Johnson D. S., Computers and intractability, W. H. Freeman and Co., San Francisco, Calif., (1979)
8. Brucker P., Scheduling algorithms (third edition), Springer-Verlag, Berlin, (2001)
9. Wolf A., Schrader G., $O(n \log n)$ Overload Checking for the Cumulative Constraint and Its Application, Lecture Notes in Computer Science, Springer, Berlin, (2006)

Development of Effective Estimation of Distribution Algorithms for Scheduling Problems

Shih-Hsin Chen · Min-Chih Chen ·
Pei-Chann Chang · Qingfu Zhang · Yuh-Min
Chen

Abstract The purpose of this paper is to establish some guidelines for designing effective Estimation of Distribution Algorithms (EDAs). These guidelines aim at balancing intensification and diversification in EDAs. Most EDAs are able to maintain some important linkages among variables. This advantage, however, may lead to the premature convergence of EDAs since the probabilistic models no longer generating diversified solutions. In addition, overfitting might occur in EDAs. This paper proposes guidelines based on the convergence speed analysis of EDAs under different computational times for designing effective EDA algorithms. The major ideas are to increase the population diversity gradually and by hybridizing EDAs with other meta-heuristics. Using these guidelines, this research further proposes an adaptive EA/G and EA/G-GA to improve the performance of EA/G. The proposed algorithm solved the single machine scheduling problems with earliness/tardiness cost in a just-in-time scheduling environment. The experimental results indicated that the Adaptive EA/G and EA/G-GA outperform ACGA and EA/G statistically significant with different stopping criteria. When it comes to the intensification of EDAs, heuristic method is combined with EDAs. Because NEH is a well-known heuristic in solving permutation flowshop problems with the objective of makespan, this heuristic generates a good initial solution for EDAs. The experimental results also indicate NEH improving the EDAs significantly. This

Shih-Hsin Chen
Department of Electronic Commerce Management, Nanhua University, Taiwan R.O.C.
E-mail: shihhsin@mail.nhu.edu.tw

Min-Chih Chen
Institute of Manufacturing Engineering, National Cheng Kung University, Taiwan R.O.C.
E-mail: cthunter@mail.wfc.edu.tw

Pei-Chann Chang
Department of Information Management, Yuan-Ze University, Taiwan R.O.C.
E-mail: iepchang@saturn.yzu.edu.tw

Qingfu Zhang
Department of Computing and Electronic Systems, University of Essex, U.K.
E-mail: qzhang@essex.ac.uk

Yuh-Min Chen
Institute of Manufacturing Engineering, National Cheng Kung University, Taiwan R.O.C.
E-mail: ymchen@mail.ncku.edu.tw

paper, hence, is of importance in the field of EDAs as well as for the researchers in studying the scheduling problems.

Keywords: Estimation of Distribution Algorithms, Intensification, Diversification, Single Machine Scheduling Problems, Permutation Flowshop Scheduling Problems

1 Introduction

In recent years, Estimation of Distribution Algorithms (EDAs) receive numerous attentions [2][7][13][29][43][63]. The procedures are to explicitly learn and build a probabilistic model to model the parental distribution, and then samples new solutions from the probabilistic model [50]. Sampling from probabilistic models avoids the disruption of partial salient genes represented by the model, contrary to what usually takes places when applying genetic operators, such as crossover and mutation operator [55]. This is the most important characteristic to distinguish EDAs and Standard Genetic Algorithms (SGA).

As claimed by [62], EDAs might be a promising method capable of capturing and manipulating the building blocks of chromosomes and, hence, efficiently solving hard optimization problems. This class of EDAs may appear as the form of cGA [29], UMDA [43], GA-EDA [51], Guided Mutation (EA/G) [63], Artificial Chromosomes with Genetic Algorithms (ACGA) [13], Self-Guided GA [20], and VNS-EDAs [55], and so on. For the detailed review, please refer to [50].

Although the previous EDAs are promising to solve hard problems when we have no knowledge about the problems, there are three cruxes of EDAs. First of all, EDAs may cause the problem of overfitting the search space and cannot represent the general information [55]. In addition, due to the premature convergence of EDAs [11], the probabilistic models no longer generate diversified solutions resulted in poor performance. Most importantly, it is still important to further improve the EDAs performance when there is knowledge can be incorporated. As a result, this paper advocates the effect the intensification and diversification in designing an effective EDAs.

This paper investigated the convergence behavior of EA/G and ACGA and some interesting results were obtained. The results shown that the convergence speed of EA/G is rather fast; however, the probabilistic models no longer generates diversified individuals after certain generations which causes a problem of premature convergence. Although ACGA may not converge fast, it outperforms the EA/G significantly when sufficient computation time is allowed. After the analysis of convergence speed, we discovered interesting connections between intensification and diversification effects of EDAs and that alternatives with other algorithms. Several guidelines were proposed and there are numerous possibilities for the researchers to develop effective searching algorithms.

In order to show the usefulness of these guidelines, the researchers proposed Adaptive EA/G, EA/G-GA, and NEH-EDAs based on the three guidelines. Two scheduling problems are employed. The first problem is the NP-Hard single machine scheduling problems with earliness/tardiness considerations in the just-in-time production environment. Adaptive EA/G and EA/G-GA are evaluated by this scheduling problem. The second scheduling problem is the minimization of makespan in the permutation flowshop scheduling problems. The general problem has been proved to be strongly NP-complete [26]. Because NEH [45] is a famous heuristic for permutation flowshop

scheduling problems, this heuristic is hybridized with some EDAs and it may benefit the EDAs to get better solution quality.

The rest of the paper is organized as follows. We review the two scheduling problems and the problem statements in Section 2 and Section 3, respectively. After that, the convergence behavior of EDAs is analyzed by using the single machine scheduling problem in Section 4 and the guidelines are deducted in Section 5. To prove the effectiveness of these proposed guidelines, this paper proposes some algorithm as examples to implement these guidelines in the same Section. Section 6 is the experimental results whereas the Adaptive EA/G and EA/G-GA are evaluated by using the single machine scheduling problems. In addition, the results of NEH with some EDAs are shown in Section 7 by testing the permutation flowshop scheduling problems. Section 8 draws the conclusions of this paper.

2 Literature Review

In order to test the performance of the proposed guidelines and algorithms, this paper employed two scheduling problems. They are the single machine scheduling problems with minimization of earliness/tardiness cost and the permutation flowshop with the objective of makespan. These scheduling review are presented in Section 2.1 and Section 2.2.

2.1 Single Machine Scheduling Problems

As a generalization of single-machine scheduling to minimize the weighted tardiness [35], the single-machine scheduling problem to minimize the total weighted earliness and tardiness costs is strongly NP-hard. The earlier works on this problem were due to [12][17][61]. [9] dealt with a similar problem with the objective of minimizing the total weighted completion time. The problem is the same as that discussed in [9]. Later on, [3] developed various dominance rules to solve the problem. [59] and [58] presented branch-and-bound algorithms based on decomposition of the problem into the weighted earliness and the weighted tardiness sub-problems. Two lower bound procedures were used for each sub-problem. The lower bound for the original problem was the sum of the lower bounds for the two sub-problems. In [59], the authors analyzed the performance of various heuristic procedures, including dispatching rules, a greedy procedure, and a decision theory based search heuristic.

The earliness/tardiness scheduling problem with equal release dates and no idle time has been considered by several authors. Both exact and heuristic approaches have been proposed to solve the problem. Among the exact algorithm approaches, branch-and-bound algorithms were presented by [1][36][38]. The lower bounding procedure of [1] is based on the sub-gradient optimization approach and the dynamic programming state-space relaxation technique, whereas [36] and [38] used Lagrangian relaxation and the multiplier adjustment method. Among these heuristics, [48] developed several dispatching rules and a filtered beam search procedure. In [58], the authors presented an additional dispatching rule and a greedy procedure. They also considered the use of dominance rules to further improve the schedule obtained by the heuristics. A neighborhood search algorithm was presented by [36].

Chang et al. [14] developed a new algorithm, termed as the Electromagnetism-like algorithm, to deal with the single-machine scheduling problem with the consideration of earliness/tardiness penalties. The Electromagnetism-like algorithm was originally proposed by [10], which is able to solve continuous problems. But the Electromagnetism-like algorithm needs to incorporate the random key method to solve discrete problems such as scheduling problems. Therefore, the Electromagnetism-like algorithm diversifies the inferior solutions, whereas the genetic algorithm operator, i.e., the crossover operator, recombines better solutions. Experimental results show that hybrid algorithms are far superior to using the Electromagnetism-like algorithm alone.

Some research has developed dominance properties (DPs) for this category of problems [38][56][40][39][57][34]. DPs are employed in branch-and-bound algorithms to enhance the fathoming procedure to be combined with other meta-heuristics, such as integrating DPs with GA to solve the scheduling problem with earliness/tardiness penalties [16].

2.2 Permutation Flowshop Scheduling Problems

In the operations research literature, flowshop scheduling is one of the most well studied problems in the area of scheduling. A permutation flowshop scheduling problem (PFSP) takes into consideration that n jobs are processed on m machines in the same order to meet one or some specified objectives. Baker [5] summarized the assumptions of permutation flowshop scheduling problems. Therefore, most of the research works emerged to develop effective heuristics and metaheuristics.

In [24], they reported a review and the classification of the heuristics for permutation flowshop scheduling problems. Hejazi and Saghaian [30] presented a complete survey of flowshop scheduling problems and contributions from 1954 to 2004. This survey considered some exact methods, constructive heuristics, metaheuristics, and evolutionary approaches. This paper is a good reference for $n/m/p/C_{max}$. In [54], they provided a comprehensive review and evaluation of permutation flowshop heuristics. Through our reading these review articles, we found it apparent that heuristics developed for PFSPs have proposed a remarkable contribution.

Heuristics are developed for some specified situations. They may not work in some unexpected situations. Among the metaheuristics, genetic algorithms have attracted a lot of attention because of many convincing results. Some research [18][52][44] did the pioneering work by applying genetic algorithms in solving flowshop scheduling problems. [53], and [65] hybridized other techniques with genetic algorithms to improve the genetic searching capability. [33] proposed an improved genetic algorithm to solve permutation flowshop scheduling problem. It is not difficult to search for more related genetic algorithm applications in the field of PFSPs. [60] considered the two-machine flowshop problems with setups in a no-wait processing environment to minimize the maximum lateness. Some dominance properties were derived which worked as a heuristic. As a result, the performance of their proposed heuristic is able to work efficiently although those applications are similar in algorithmic structures. Leaving the traditional structures behind, this work intends to improve the effectiveness and efficiency of genetic search by embedding more feedback information in the evolutionary process. For the more detailed information about the review of the flowshop scheduling, please refer to Ruiz and Maroto [54] who did an extensive comparison in the flowshop

scheduling problems, including tabu search, simulated annealing, genetic algorithms, iterated local search and hybrid techniques.

3 Problem Statements

3.1 Statements of Single Machine Scheduling Problems

In this paper, a deterministic single machine scheduling problem without release date is investigated and the objective is to minimize the total sum of earliness and tardiness penalties. A detailed formulation of the problem is described as follows: A set of n independent jobs $\{J_1, J_2, \dots, J_n\}$ has to be scheduled without preemptions on a single machine that can handle one job at a time at most. The machine is assumed to be continuously available from time zero onwards and unforced machine idle time is not allowed. Job $J_j, j = 1, 2, \dots, n$ becomes available for the processing at the beginning, requires a processing time p_j and should be completed on its due date d_j . For any given schedule, the earliness and tardiness of J_j can be respectively defined as $E_j = \max(0, d_j - C_j)$ and $T_j = \max(0, C_j - d_j)$, where C_j is the completion time of J_j .

The objective is then to find a schedule that minimizes the sum of the earliness and tardiness penalties of all jobs $\sum_{j=1}^n (\alpha_j E_j + \beta_j T_j)$ where α_j and β_j are the earliness and tardiness penalties of job J_j . The inclusion of both earliness and tardiness costs in the objective function is compatible with the philosophy of just-in-time production, which emphasizes producing goods only when they are needed. The early cost may represent than generating cost of completing a product early, the deterioration cost for perishable goods or a holding (stock) cost for finished goods. The tardy cost can represent rush shipping costs, lost sales and loss of goodwill. It is assumed that no unforced machine idle time is allowed, so the machine is only idle when no job is currently available for processing. This assumption reflects a production setting where the cost of machine idleness is higher than the early cost stemming from completing any job before its due date, or the capacity of the machine is limited when compared with its demand, so that the machine must be kept running all the time. Some specific examples of production arrangements with these characteristics are provided by [48] and [61]. The set of jobs is assumed to be ready to process jobs at the beginning which is a characteristic of the deterministic problem.

3.2 Formulation of Permutation Flowshop Scheduling Problems

Flowshops are useful tools in modeling manufacturing processes. A permutation flowshop is a job processing facility, which consists of several machines and several jobs to be processed on the machines on different machines. In a permutation flowshop, all jobs follow the same processing order. Our objective is to find a set of compromise solutions so that the makespan is minimized.

The flowshop scheduling problem is a typical assembly line problem where n different jobs have to be processed on m different machines. All jobs are processed on all the machines in the same order. The processing time of the jobs on machines are fixed regardless of the order in which the processing is conducted. The problem is characterized by a matrix $P = (p_{ij}), i = 1, \dots, n, j = 1, \dots, m$, of processing time. Each machine processes exactly one job at a time and each job is processed on exactly one machine

at a time. The problem then is to find a sequence of jobs of minimizing the makespan which is the completion time of the last job in the sequence on the last machine.

If C_i denotes the completion time for job i , we are trying to minimize $\max C_i$. There are many other criteria that can be considered for the purpose of optimization. We refer the reader to [4] for a detailed discussion of scheduling using GA. For details of the flowshop and other scheduling and sequencing problems we refer the reader to [5].

The flowshop scheduling can be formerly defined as follows: if $p(i, j)$ is the processing time for Job i on Machine j , and a job permutation $\{\pi_1, \pi_2, \dots, \pi_n\}$, where there are n jobs and m machines, accordingly the completion times $C(\pi_i, j)$ is calculated as follows:

$$C(\pi_1, 1) = p(\pi_1, 1) \quad (1)$$

$$C(\pi_i, 1) = C(\pi_{i-1}, 1) + p(\pi_i, 1) \text{ for } i = 2, \dots, n \quad (2)$$

$$C(\pi_1, j) = C(\pi_1, j-1) + p(\pi_1, j) \text{ for } j = 2, \dots, m \quad (3)$$

$$C(\pi_i, j) = \max\{C(\pi_{i-1}, j), C(\pi_i, j-1)\} + p(\pi_i, j) \quad (4)$$

$$\text{for } i = 2, \dots, n; j = 2, \dots, m \quad (5)$$

The makespan is finally defined as

$$C_{\max}(\pi) = C(\pi_n, m). \quad (6)$$

Subsequently, the objective is to find a permutation π^* in the set of all permutations Π so that

$$C_{\max}(\pi^*) \leq C_{\max}(\pi) \quad \forall \pi \in \Pi. \quad (7)$$

A more general flowshop scheduling problem can be defined by allowing the permutation of jobs to be different on each machine. However, what work has been done to show on the more general flow shop scheduling problem has tended to small improvement in solution quality over the permutation flowshop scheduling problems (PFSP) while increasing the complexity of the problem substantially. The size of the solution space increases from $n!$ to $(n!)^m$. Other objective functions for the PFSP also received a lot of attention. For example, the mean flow-time (the time a job spends in the process), or the mean tardiness (assuming some deadline for each job) are to be minimized. Other real problems from the manufacturing industries such as their jobs may have non-identical release dates, and there may be sequence-dependent setup times, and limited buffer storage between machines and so on. These characteristics of the real world problems will make the problem more complicated to be solved within a reasonable time frame. However, GA approaches provide a more realistic view to the problem. Since it can generate alternatives of sequences (in the evolving process, each chromosome represents a feasible solution to the problem) to the decision maker, a more applicable sequence can be decided to solve the current problem with satisfactory results.

4 Convergence Behavior Analysis of EDAs

In this section, ACGA and EA/G are analyzed by running the instances of single machine scheduling problems with the minimization of earliness/tardiness cost. When we observe the convergence progress of the two EDAs, Standard Genetic Algorithm (SGA) with elitism is served as a base-line into the comparisons. The stopping criteria are based on the number of examined solutions, which are 50000, 75000, 100000, and 125000 solutions. The four examined solutions stand for the different implementation environments which allow lower, medium, high, and higher level of computational time. When taking a close look of the convergency behavior, we attempt to discover whether there is any difference among the three algorithms when the stopping criteria are different. The analysis is done by Design-of-Experiment to distinguish the difference of these algorithms. And the parameter settings are the same, such as the population size is 100, the crossover rate is 0.9, and mutation rate is 0.5 across all experiments. And when we use 50000 solutions, it means that the algorithms stop at generation 500.

There are numerous data sets published in the literature [56] for the single machine scheduling problems, including 20, 30, 40, 50, 60, and 90 jobs. Each data set of 20 jobs up to 50 jobs contains 49 instances (problems) whereas there are 9 instances in the data set of 60 jobs and 90 jobs. We carried out our experiments on these total 214 instances. Each algorithm will replicate every instance 30 times.

In order to test the significance among the compared algorithms which under different stopping criteria, Analysis of Variance (ANOVA) is used. When the P-Value is less than 0.05, it means there is a significance of the factor. From Table 1 to Table 4, the statistic model which considers method, instances, and the interaction between the method and instances (method*instances). The factor, method, is very significant in the all ANOVA tables, Duncan Grouping test is used to further distinguish the performance of the two algorithms. In Duncan Group test, when the algorithms share the same alphabet, it means they are in the same group so that there is no difference between/among these algorithms. On the other hand, as soon as they are not in the same group (or to share the same alphabet), there is significant difference between/among them [42].

Table 1 ANOVA results of comparing EA/G and ACGA: 50000 examined solutions

Source	DF	SS	Mean Square	F Value	P Value
instances	213	2.66E+12	12499046967	5568744	<.0001
method	1	452571.0981	452571.0981	201.64	<.0001
instances*method	213	14622265.17	68649.13225	30.59	<.0001
Error	12412	27858737.01	2244.500243		
Corrected Total	12839	2.66E+12			

The Duncan grouping results are from Table 5 to Table 8. The tables show that when EA/G outperforms the ACGA under the stopping criterion of using 50000 and 75000 solutions. There is no difference between EA/G and ACGA when they both apply 100000 solutions. Finally, ACGA outperforms the EA/G statistically significant in the case of employing 125000 solutions. A performance transition is occurred at the stopping criterion of using 100000 solutions.

In order to show the results clearly, Fig. 1 which utilizes the instance sks952a which may support for the scenario of performance transition. EA/G converges faster than

Table 2 ANOVA results of comparing EA/G and ACGA: 75000 examined solutions

Source	DF	SS	Mean Square	F Value	P Value
instances	213	2.66E+12	12474639403	7114206	<.0001
method	1	6833.386994	6833.386994	3.9	0.0484
instances*method	213	1405887.963	6600.412972	3.76	<.0001
Error	12412	21764231.71	1753.483058		
Corrected Total	12839	2.66E+12			

Table 3 ANOVA results of comparing EA/G and ACGA: 100000 examined solutions

Source	DF	SS	Mean Square	F Value	P Value
instances	213	2.66E+12	12469679239	7587469	<.0001
method	1	474.7633178	474.7633178	0.29	0.5909
instances*method	213	608977.2534	2859.048138	1.74	<.0001
Error	12412	20398589.78	1643.457121		
Corrected Total	12839	2.66E+12			

Table 4 ANOVA results of comparing EA/G and ACGA: 125000 examined solutions

Source	DF	SS	Mean Square	F Value	P Value
instances	213	2.66E+12	12467759681	8878547	<.0001
method	1	8858.465846	8858.465846	6.31	0.012
instances*method	213	506457.0649	2377.732699	1.69	<.0001
Error	12412	17429635.63	1404.256818		
Corrected Total	12839	2.66E+12			

Table 5 Duncan grouping of comparing EA/G and ACGA: 50000 examined solutions

Duncan Grouping	Mean	N	Method
A	12826.724	6420	ACGA
B	12814.85	6420	EA/G

Table 6 Duncan grouping of comparing EA/G and ACGA: 75000 examined solutions

Duncan Grouping	Mean	N	Method
A	12814.902	6420	ACGA
B	12813.443	6420	EA/G

Table 7 Duncan grouping of comparing EA/G and ACGA: 100000 examined solutions

Duncan Grouping	Mean	N	Method
A	12813.66	6420	EA/G
A			
A	12813.276	6420	ACGA

Table 8 Duncan grouping of comparing EA/G and ACGA: 125000 examined solutions

Duncan Grouping	Mean	N	Method
A	12812.898	6420	EA/G
B	12810.149	6420	ACGA

ACGA and SGA; however, the performance of EA/G doesn't be improved with the number of examined solutions increased after the level of 75000. After the generation 150, EA/G is converged and the performance is not improved. ACGA, nonetheless, is improved generation by generation and this algorithm is superior to EA/G when we apply longer computational time. As a result, it could be a problem of premature convergence belonged to EA/G. So we may increase the diversity of the generated solutions for the EA/G or the EDAs completely sample new individuals from probabilistic models.

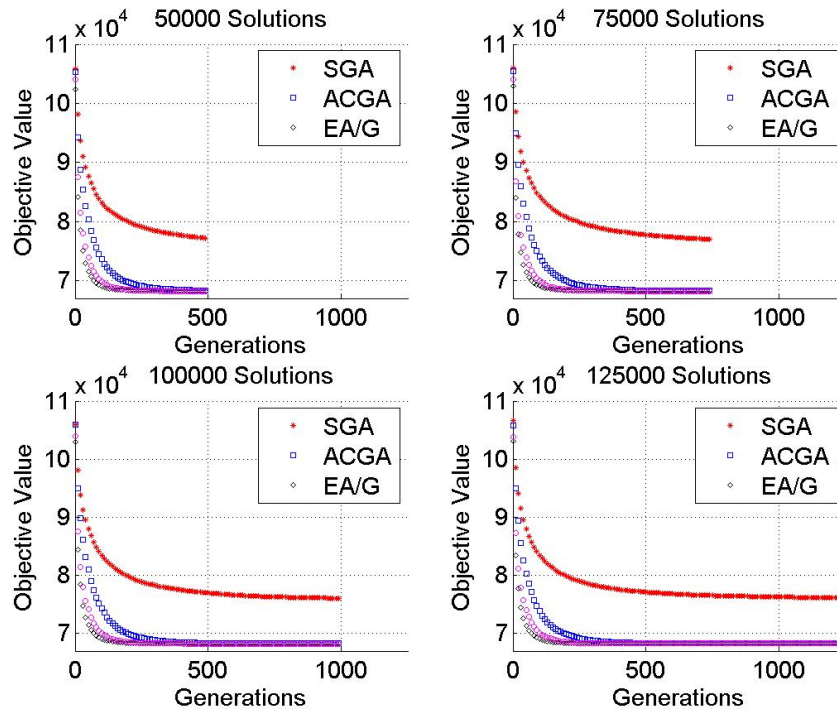


Fig. 1 convergence analysis of the algorithms in different stopping criteria (instance-sks952a)

To conclude the comparison results, these experiments reveal interesting points when the three algorithms are ran under the various stopping criteria. EA/G outperforms the ACGA statistically significant when it stops under lower computational time whereas ACGA performs well when we apply higher level of stopping criterion. It shows EA/G might converge faster than ACGA; however, when we concern on the solution quality and we are able to employ higher level computing time. As a result, this paper continually discusses this phenomenon and then conduct some important guidelines in Section 5.

5 Guidelines of Designing Effective EDAs

There are some interesting results obtained in [21]. EA/G is better than ACGA when the computational time is limited. On the other hand, ACGA outperforms EA/G when there is enough computational time for the algorithm to be fully converged. The reason is because sampling new solutions from probabilistic models avoids the disruption of good solution structures represented by the model. During the early evolutionary process, there are some salient genes that are available. Since EA/G completely samples new solutions from probabilistic models, it has the advantage of fast convergency in the early stage as shown in the convergency plots of the previous section. However, EA/G seems no longer producing better solutions after 200 generations or so in most instances. The early convergence effect is quite obvious in EA/G after some generations are executed.

Contrary to EA/G, the convergency speed of ACGA is not that fast in the early stage. ACGA, nevertheless, performs well in the case that the solution quality is more important as that the computational times are long enough to sustain. The implementation time can be controlled in a reasonable range as soon as the pre-determined solution quality is reached. ACGA is a hybrid algorithm integrated by probabilistic models and SGA. This hybrid algorithm samples new solutions from probabilistic models after certain periods of SGA runs during the evolutionary process. The implication of this framework means that ACGA may converge slower because the genetic operators may disrupt the structure of good solutions once a while. As for the advantage of genetic operators, it generates better population diversity due to the disruptions by genetic operators. The intensification and diversification effect present the trade-off in the searching progress of ACGA.

As observed in the ACGA and EA/G evolutionary procedures, there should be a systematic way in balancing the intensification and diversification of a Meta-heuristic. For instance, Reactive Tabu Search [8][47] diversifies the solution by making a number of random moves when the search appears to be repeating an excessive number of solutions. Under this assumption, ACGA has better chances to explore different solution space than EA/G with a longer computational time. Due to these interesting phenomena, we propose some specific guidelines for developing an effective searching algorithm in the following sections. In order to verify the effectiveness of these guidelines, this paper proposed an Adaptive EA/G, EA/G-GA, and NEH-EA/G and NEH-ACGA which utilize these guidelines proposed to enhance the solution quality of the EDAs significantly for different scheduling problems. The proposed algorithms are shown in the end of the sections.

5.1 Increasing the Population Diversity Gradually

The stagnation of EA/G apparently exists after some generation runs. It could be a common problem for other EDAs which entirely sample new solutions from probabilistic models. For example, PBIL [6], UMDA [41], cGA [27], BMDA [46], ECGA [28], BOA [48] are all categorized in this class. As a result, a new design of these EDAs shall consider the early convergence effect and try to increase population diversity when the population no longer generates diversified or better solutions.

In terms of revising these existing EDAs, a key approach is to control the probabilistic models after a number of generation runs. Some diversity preservation techniques

are proposed in [11], which stem from the concept of Ant-Colony Optimization [22][25]. They test the diversity preservation techniques in solving the single machine scheduling problems and flowshop scheduling problems. Their results indicate the best-objective evaporation and Max-Min evaporation performs significantly better for these scheduling problems. The same principle is applicable for all other EDAs.

Apart from the general methodology of controlling the probabilistic models, the other method is to modify each algorithm based on their own problem characteristics. Take EA/G for example, β is the parameter which determines a proportion of genes replicated from a selected elite solution. β is set as a static parameter in the original version of EA/G. In order to increase the diversity, β can be decreased generation by generation so that the diversity of the population will be increased. In order to explain the ideas of these guidelines clearly, this paper will design an Adaptive EA/G by changing the parameter of β in Section 5.4.

5.2 EDAs Alternative with Other Meta-heuristics

Since EDAs may not generate diversify solutions well and cause the problem of premature convergency, incorporation of other meta-heuristic, particularly the genetic algorithm, might be a good approach to raise the the population diversity. There are manifold existing hybrid frameworks of combining the EDAs with genetic algorithms, such as GA-EDA, Model-Based EA, ACGA and Self-Guided GA. The main characteristics of these algorithm is to alternate EDAs together with genetic algorithms. GA-EDA and Model-Based EA apply the probabilistic models and genetic operators one after the other. ACGA utilizes the probabilistic models occasionally. Self-Guided GA is different from all EDAs because it does not sample new solutions from the probabilistic models. The probabilistic models, instead, are treated as a fitness surrogate to evaluate the fitness of new solutions beforehand in each generation. Thus these hybrid algorithms could yield better solution quality when longer computational efforts are allowed.

It is clearly that GA-EDA, ACGA, and Model-Based EA alternate the EDAs with GAs. The solution quality of these algorithms can be further enhanced by implementing the searching strategy of intensification using the EDAs and diversification via GAs. Since EDAs may converge faster in the beginning and genetic operators might provide better population diversity, EDAs could be implemented frequently in the early stage while GAs dominate in the later period. As a result, we can modify GA-EDA and Model-Based EA by increasing the frequency of the EDAs in the early stage and then decrease the usage of EDAs in the later stage. In addition, ACGA *injects* artificial chromosomes into the population periodically, the new version of ACGA could inject the artificial chromosome more frequently in the earlier stage and then slowly decrease the injection of the artificial chromosome in the later stage.

Except generating a population of chromosomes by probabilistic models or genetic operators in a single generation, a possible way is to sample a proportional solutions from probabilistic models and the rest of solutions are generated by using crossover and mutation operator. MGSPGA [15] samples 20% solutions from a probability matrix and the rest of the chromosomes are generated by the genetic operators. Consequently, the goal of intensification and diversification are balanced in each generation. Because it is still unknown about which approach is better (samples new solutions entirely or partially), researchers should examine the two strategies in the near future.

Although most previous researchers employ genetic algorithms, the authors advocate that it is not limited to alternate EDAs with genetic algorithms. EDAs could work with other meta-heuristics. For example, an artificial immune system (AIS) might be able to generate diversified solutions [49]. It is because an AIS employs many properties of natural immune systems, particularly the diversity [32]. This natural system possesses a content addressable memory and the ability to forget little-used information [31]. Thus, AIS may provide diverse solutions that result in good solution quality when we alternate the EDAs with AIS.

Finally, there is a benefit of not sampling any solution from probabilistic models. When we solve sequential problems, the time-complexity of sampling procedures by applying the proportional selection is $O(n^2)$. It is particularly time-consuming when we solve larger size problems. As a result, we should use more efficient approach without using the proportional selection. The next section describes the methods to solve the crux of EDAs in solving the sequential problems by using the concept of Self-Guided GA.

5.3 Incorporate EDAs with Other Heuristics for Scheduling Problems

Although Zhang et al. [64] point out the positive effect of combining other heuristics and it may bring better solution quality, the meanings are two-fold in this paper. First of all, there are many possibilities to adopt some heuristics with EDAs and heuristic could enhance the convergency effect. Take the scheduling problems for instance, dominance properties are the mathematical derivations for the scheduling problems. Chen et al. [19] employ this technique to work with an estimation of distribution algorithm. As a result, the performance of this hybrid algorithm is better than any previous algorithms which work separately.

When we solve the flowshop scheduling problems which minimize the makespan, there is a famous heuristic which is named NEH [45]. This heuristic is used to generate a good initial solution and then the meta-heuristic is used to further improve the solution quality. This paper examines the NEH heuristic with different EDAs and the experimental result shown that NEH improves the performance of EDAs significantly.

Finally, because the heuristic method is used to generate good solutions, the researchers should pay more attentions to the diversity issue of EDAs. The reason is before EDAs carry out the exploration, there is a good basis provided by heuristics. It is apparently that the designing strategy should consider how to let the EDAs yield better population diversity. A heuristic method, consequently, may work well with ACGA because a heuristic method is responsible for the convergency and ACGA produces diversified population. In addition, according to the guideline one, EA/G may apply lower β value which reduces the number of copied genes from a selected elite individual when combining with a heuristic. In general, EDAs work together with a heuristic is a good approach because we may obtain better solution quality in a shorter time and we should avoid probabilistic models no longer generate diversified individuals.

5.4 Implementation of First Guideline: Adaptive EA/G

To prove usefulness of the guidelines, this paper implements the first guideline which increases the population diversity gradually. Because other EDAs may have the problem

S : A set of shuffled sequence which determines the sequence of each position is assigned a job.
 Ω : is the set of un-arranged jobs.
 J : The set of arranged jobs. J is empty in the beginning.
 θ : A random probability is drawn from $U(0, 1)$.
 i : A selected job by proportional selection
 k : The element index of the set S

```

1:  $S \leftarrow$  shuffled the job number  $[1 \dots n]$ 
2:  $J \leftarrow \Phi$ 
3: Calculating  $\beta' = \beta * (MaxGen - g) / (MaxGen)$ 
4: Guided mutation Operator  $y = GM(p, x, \beta')$   $\beta' \in [0, 1]$ .
5: Output:  $y = (y_1, \dots, y_n) \in \{0, 1\}^n$ .
6: while  $k \neq \Phi$  do
7:   Flip a coin with head probability  $\beta'$ ;
8:   if The head turns up then
9:     Select a job  $i$  satisfies  $\theta \leq P_{ik} / \sum_{i \in \Omega} P(i, k)$ 
10:     $J(k) \leftarrow i$ 
11:   else
12:     $y_i := x_i$ .
13:   end if
14:    $\Omega \leftarrow \Omega \setminus i$ 
15:    $S \leftarrow S \setminus k$ 
16: end while

```

Fig. 2 Adaptive EA/G: MainProcedure()

of premature convergency, this section illustrates the potential implications of the same class of EDAs.

In EA/G, the parameter β is an important parameter which controls the percentage of copied genes from an elite solution. In our previous research [20], this parameter is statistically significant in solving the single machine and flowshop scheduling problems. The reason is if β is higher, it means EA/G converges faster in the beginning. In addition, we discuss that the EA/G no longer improves the solution quality after 100 or 200 generations in most instances in guideline 1. The convergency progress apparently reveals the stagnation of this algorithm and the main reason might be the population diversity decreased. As a result, this paper proposes an Adaptive EA/G by adjusting the parameter β generation by generation.

We define the total number of generation is as $MaxGen$ and g is the current generation. In Eq. 8, it shows that the new setting of β as β' . Thus in the very beginning, the β' is high and the β is decreased gradually with the evolutionary progress. Through this simple approach, the β' is decreased and the number of copied elite genes are decreased so that the population diversity is increased.

$$\beta' = \beta * (MaxGen - g) / (MaxGen) \quad (8)$$

Fig. 5.4 demonstrates the main procedures of the Adaptive EA/G. The differences are in Step 1, Step 2, and Step 6 which calculate the β' first and then let the EA/G to apply this new β' value. In order to validate the performance of the Adaptive EA/G, it is thus compared with other algorithms and the experimental result is shown in the Section 6 in solving the single machine scheduling problems under different stopping criteria. The following is the pseudo code of the proposed Adaptive EA/G.

pop: The whole population.
 β : The proportional of copied genes from an elite solution
t: Current Generation

```

1:  $t \leftarrow 0$ 
2: Initialize  $P(t)$ 
3: while  $t < generations$  do
4:   if  $t \% 2 == 0$  then
5:     GuidedMutationOperator(pop,  $\beta$ )
6:   else
7:     GA(pop).
8:   end if
9:    $t \leftarrow t + 1$ 
10: end while

```

Fig. 3 EA/G-GA: MainProcedure()

5.5 Implementation of Second Guideline: EA/G-GA

There are two reasons why EDAs may alternate with other metaheuristics. First of all, when we solve sequencing problems and n is the problem size, the time-complexity of EDAs sampling a new solution from probability models needs $O(n^2)$. It is time-consuming for larger size problems even though EDAs bring better solution quality. When it works with other metaheuristic, EDAs will work more efficiently. In addition, other metaheuristic might introduce new solutions into the population so that the diversity is maintained. It avoids the premature convergency of EDAs. This paper combines the EA/G with Standard Genetic Algorithm (SGA) as an example, which is named EA/G-GA.

The main characteristic of the EA/G-GA is to alternate the EA/G and GA by turn. Once the current generation is done by EA/G, GA carries out the evolution progress in the next generation. Through the alternation of the EA/G and GA, the researchers expect the computing efforts will be reduced and the premature convergency of EA/G is avoided because the diversity is improved. The following figure is the pseudo code of the EA/G-GA.

5.6 Implementation of Third Guideline: NEH heuristic for EDAs in Solving Flowshop Scheduling Problems

Due to flowshop is a strong NP-Hard problem, many researchers employed the NEH to generate a good initial solution for metaheuristics. This research integrates the NEH with some EDAs into together, such as EA/G and ACGA. The proposed are named NEH-EA/G and NEH-ACGA when NEH works with EA/G and ACGA, respectively. As a result, except for the generation of initial solutions are slightly different (one solution is obtained from NEH and the remaining the solutions are generated randomly), the rest of procedures of EA/G and ACGA are the same. The procedures of NEH is shown below.

Some researchers shown the improved version of NEH [23][37]. As a result, the implementation result might be further enhanced by using these improved NEH methods.

k : The sorting index of jobs while $k=1, \dots, n$

- 1: Order the n jobs by decreasing sums of processing times on the machines
- 2: Take the first two jobs ($k = 1$ and $k = 2$) and minimize the makespan of the two jobs
- 3: for the following k th jobs ($k=3, \dots, n$), insert it into k possible ones, which minimizes the partial makespan.

Fig. 4 NEH: MainProcedure()

6 Experiment Results of Adaptive EA/G and EA/G-GA for Single Machine Scheduling Problems

Before we demonstrate the experiment results, we explain the environment settings, including the problem instances, parameter configurations, and analysis methods.

6.1 Settings of the Experiments

All the algorithms are analyzed by running the instances of single machine scheduling problems with the minimization of earliness/tardiness cost. It is the consideration of just-in-time production environment. There are numerous data sets published in the literature [56] for the single machine scheduling problems, including 20, 30, 40, 50, 60, and 90 jobs. Each data set of 20 jobs up to 50 jobs contains 49 instances (problems) whereas there are 9 instances in the data set of 60 jobs and 90 jobs. We carried out our experiments on these total 214 instances. Each algorithm will replicate every instance 30 times.

The stopping criteria are based on the number of examined solutions, which are 50000, 75000, 100000, and 125000 solutions, which is identical to our previous research in [21]. The four solution evaluations stand for the different implementation environments which allow lower, medium, high, and higher level of computational time. And the parameter settings are the same, such as the population size is 100, the crossover rate is 0.9, and mutation rate is 0.5 across all experiments. And when we use 50000 solutions, it means that the algorithms stop at generation 500.

The analysis is done by Design-of-Experiment to distinguish the difference of these algorithms. The comparison method uses the ANOVA. When the P-Value of a factor in the ANOVA table is less than 0.05, it means there is a significance of this factor. Once a factor Method is significant, Duncan Grouping test is used to further distinguish the performance of the levels belonged to the factor. In Duncan Group test, when the algorithms share the same alphabet, it means they are in the same group so that there is no difference between/among these algorithms. On the other hand, as soon as they are not in the same group (or to share the same alphabet), there is significant difference between/among them. The detailed explanations of statistics terms are available in [42].

Finally, because of the page limits, the complete experimental results and the technical report [21] are available on our website ¹. The following subsections are the empirical results of solving the single machine scheduling problems.

¹ <http://mail.nhu.edu.tw/~shihhsin/publications/sourceCodes/convergencyAnalysis/>

6.2 Empirical Results

The proposed algorithms Adaptive EA/G and EA/G-GA are compared with the original EA/G, ACGA, and SGA. They are tested under various stopping criteria. These algorithms evaluate 50000, 75000, 100000, and 125000 solutions. The comparison method still uses the ANOVA which is mentioned in Section 4 and the explanations of statistics terms are available in [42]. The ANOVA Table 9 to Table 12 which show the significance of these compared algorithms due to the P-Value is less than 0.0001. Thus a Duncan grouping analysis is used to compare the significance of among the four algorithms.

Table 9 ANOVA results at the stopping criterion of 50000 examined solutions

Source	DF	SS	Mean Square	F Value	P Value
instances	213	5.32E+12	24966564338	1.41E+07	<.0001
method	3	990458.8414	330152.9471	186.73	<.0001
instances*method	639	21834746.74	34170.1827	19.33	<.0001
Error	24824	43891753.69	1768.117696		
Corrected Total	25679	5.32E+12			

Table 10 ANOVA results at the stopping criterion of 75000 examined solutions

Source	DF	SS	Mean Square	F Value	P Value
instances	213	5.31E+12	24941096330	1.64E+07	<.0001
method	3	107158.5288	35719.50961	23.43	<.0001
instances*method	639	2569676.571	4021.403085	2.64	<.0001
Error	24824	37840343.96	1524.345148		
Corrected Total	25679	5.31E+12			

Table 11 ANOVA results at the stopping criterion of 100000 examined solutions

Source	DF	SS	Mean Square	F Value	P Value
instances	213	5.31E+12	24936063497	1.74E+07	<.0001
method	3	92494.17975	30831.39325	21.5	<.0001
instances*method	639	1350141.22	2112.897058	1.47	<.0001
Error	24824	35594800.28	1433.886573		
Corrected Total	25679	5.31E+12			

Table 12 ANOVA results at the stopping criterion of 125000 examined solutions

Source	DF	SS	Mean Square	F Value	P Value
instances	213	5.31E+12	24931260598	1.93E+07	<.0001
method	3	63931.9233	21310.6411	16.5	<.0001
instances*method	639	1460446.674	2285.519051	1.77	<.0001
Error	24824	32070591.19	1291.918756		
Corrected Total	25679	5.31E+12			

In Duncan analysis, when the algorithms share the same alphabet, there is no difference between/among the algorithms [42]. On the other hand, if the algorithms don't share the same alphabet, they are different to each other. Thus in Table 13 to Table 16 present the results of the comparisons. EA/G-GA and Adaptive EA/G are consistently significant across the different stopping criteria compared to the original EA/G. Take the convergence plot and the average objective value plot (See Fig. 5 and Fig. 6) for example, the solution quality of EA/G-GA and Adaptive EA/G are better than ACGA and SGA. The reason is that EA/G-GA and Adaptive EA/G maintain better population diversity, they outperform the original EA/G in the end of the generations. In addition, EA/G-GA is significantly better than Adaptive EA/G. The reason might be the genetic operators which produce more diversified chromosomes than we just change the parameter of the algorithm. The implication means the more diversity is, the better performance we can have. Thus it is an important direction for the researchers.

When it comes to the comparison between Adaptive EA/G and ACGA, Adaptive EA/G is better than ACGA through 50000 to 100000 except in the stopping criterion of using 125000 solutions. Adaptive EA/G and ACGA works equally when the 125000 solutions are used. The reason of Adaptive EA/G doesn't outperform the ACGA is because the ACGA keeps better population diversity so that this algorithm has better chance to find out better solution in the longer computational time. As a result, when we allow to spend more computational time, we can increase the diversity of the Adaptive EA/G being higher. So the proposed algorithm Adaptive EA/G is robust when the algorithm is used in different circumstances.

Table 13 Duncan grouping at the stopping criterion of 50000 examined solutions

Duncan Grouping	Mean	N	Methods
A	12826.724	6420	ACGA
B	12814.85	6420	EA/G
C	12812.509	6420	Adaptive EA/G
D	12810.899	6420	EA/G-SGA

Table 14 Duncan grouping at the stopping criterion of 75000 examined solutions

Duncan Grouping	Mean	N	Method
A	12814.902	6420	ACGA
B	12813.531	6420	EA/G
C	12812.059	6420	Adaptive EA/G
D	12809.391	6420	EA/G-SGA

Finally, Table 17 lists the basic statistic results of Adaptive EA/G and EA/G-SGA under different solution evaluations.

Table 15 Duncan grouping at the stopping criterion of 100000 examined solutions

Duncan Grouping	Mean	N	Method
A	12813.66	6420	EA/G
A	12813.276	6420	ACGA
B	12811.168	6420	Adaptive EA/G
C	12808.906	6420	EA/G-SGA

Table 16 Duncan grouping at the stopping criterion of 125000 examined solutions

Duncan Grouping	Mean	N	Method
A	12812.898	6420	EA/G
B	12810.827	6420	Adaptive EA/G
B	12810.149	6420	ACGA
C	12808.48	6420	EA/G-SGA

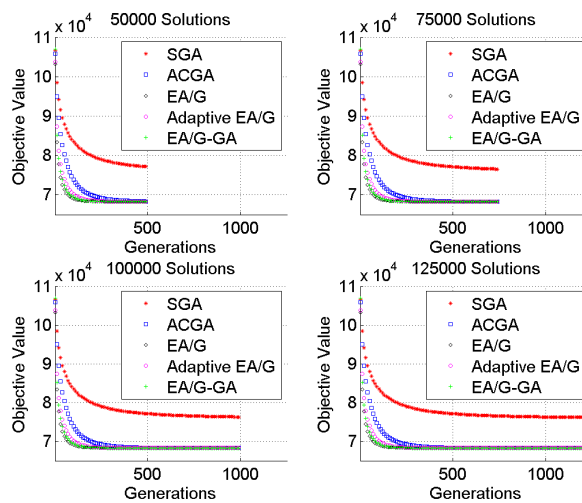


Fig. 5 Compare the convergency of Adaptive EA/G and EA/G-SGA with other algorithms

7 Experiment Results of NEH-EDAs for Permutation Flowshop Scheduling Problems

The permutation flowshop scheduling problems are to minimize the makespan of n jobs on m machines. The testing instances are drawn from OR-Library². NEH-EDAs, including the NEH-EA/G and NEH-ACGA, are compared with SGA, Mining Gene GA (MGGA) and ACGA in [13], and EA/G [63]. Each combination is replicated 30 times and every algorithm examines the instances by 30000 solutions. Table 18 is the ANOVA result of the method comparisons. Since the factor (method) presents significant difference, the Duncan pair-wise test is further employed in Table 19.

² <http://people.brunel.ac.uk/~mastjjb/jeb/info.html>

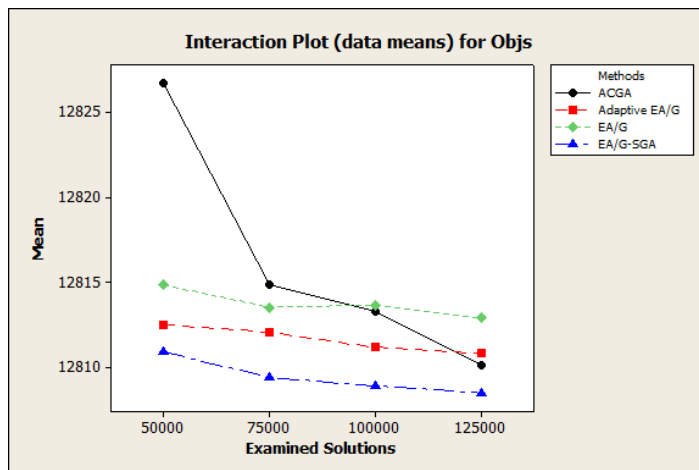


Fig. 6 Average Objective values under different stopping criteria

In these comparisons, when the NEH works with EA/G and ACGA, the performance of the two algorithms are improved significantly. As a result, the performance of the EDAs might be improved when an effective heuristic is used. The following Table 20 is the selected results of the permutation flowshop scheduling problems.

8 Conclusions

This paper analyzed the convergence behavior of some EDAs. The experiment results shown the facts of intensification and diversification effect of EDAs. We compare EA/G with ACGA and SGA. EA/G completely samples new individuals from probabilistic models while the ACGA generates new solution by using the probabilistic models periodically. In the fact that the EDAs may not disrupt the good solution structure, the convergency speed is faster than that of using genetic operators. That is why EA/G converges faster than ACGA. However, due to the premature convergency of probabilistic models, EA/G no longer generates diversified solutions so that it is not able to get better solution quality. On the other hand, ACGA converges slower than EA/G but when we extend the computational time, the performance is improved. Thus it reveals interesting points of the convergency analysis. Based on these guidelines, some guidelines are deducted for designing effective EDAs.

The guidelines include (1) Increasing population diversity gradually (2) Hybridization of EDAs with Other Meta-heuristics (3) Incorporate EDAs with Other Heuristics. The first guideline is very important for all EDAs because they may encounter this problem of premature convergency. There are some solving approaches are given and an Adaptive EA/G is proposed. Adaptive EA/G further improves the solution quality statistically significant.

In guideline 2, the reason of incorporating EDAs with other meta-heuristics is to take the advantage of meta-heuristic which is able to provide better diversity. The approach is definitely different from the guideline 1 because the metaheuristic provides more diversified solutions than we just change the parameters of an algorithm. The experimental results indicated EA/G-GA outperforms the Adaptive EA/G significantly across different. As a result, it is an important direction for all EDAs.

The third guideline emphasizes on the usefulness when we have prior knowledge of the problem. Because NEH is a good heuristic when we like to tackle with the permutation flowshop scheduling problems, it is thus integrated with EDAs. The results shown that NEH helps the EA/G and ACGA statistical significantly. So a good heuristic might be necessary when we solve a large size problem in practice. Finally, we still advocate we should be cautious about the solution diversity when we incorporate some heuristics. The reason is that heuristic methods are capable of providing better intensification effects; however, the population diversity is decreased in the same time. As a result, the EDAs may increase the diversity of the solutions by using the guideline 1 or the guideline 2 which might enhance the solution quality.

For the future research, researchers may test the guidelines to solve various problems and then to validate these guidelines.

Acknowledgements The authors appreciate the supports from the National Science Council, Taiwan, R.O.C. with grant contract No. NSC 98-2218-E-343-001-MY2.

References

1. Abdul-Razaq, T., Potts, C.: Dynamic Programming State-Space Relaxation for Single-Machine Scheduling. *The Journal of the Operational Research Society* **39**(2), 141–152 (1988)

2. Aickelin, U., Burke, E.K., Li, J.: An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. *Journal of the Operational Research Society* **58**(12), 1574–1585 (2007)
3. Alves, M.J., Almeida, M.: MOTGA: A multiobjective Tchebycheff based genetic algorithm for the multidimensional knapsack problem. *Computers and Operations Research* **34**(11), 3458–3470 (2007)
4. Bagchi, T.P.: *Multiobjective Scheduling by Genetic Algorithms*. Springer (1999)
5. Baker, K.R.: *Introduction to sequencing and scheduling*. Wiley (1974)
6. Baluja, S., Davies, S.: Using Optimal Dependency-Trees for Combinational Optimization. *Proceedings of the Fourteenth International Conference on Machine Learning table of contents* pp. 30–38 (1997)
7. Baraglia, R., Hidalgo, J., Perego, R., CNUCE, I., CNR, P.: A hybrid heuristic for the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on* **5**(6), 613–622 (2001)
8. Battiti, R., Tecchiolli, G.: The Reactive Tabu Search. *ORSA JOURNAL ON COMPUTING* **6**, 126–126 (1994)
9. Belouadah, H., Posner, M., Potts, C.: Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete Applied Mathematics* **36**(3), 213–231 (1992)
10. Birbil, S., Fang, S.C.: An Electromagnetism-like Mechanism for Global Optimization. *Journal of Global Optimization* **25**(3), 263–282 (2003)
11. Chang, P., Chen, S.H., Fan, C.Y.: Generating Artificial Chromosomes by Mining Gene Structures with Diversity Preservation for Scheduling Problems. accepted by *Annals of Operations Research* (2009)
12. Chang, P.C.: A branch and bound approach for single machine scheduling with earliness and tardiness penalties. *Computers and Mathematics with Applications* **37**(10), 133–144 (1999)
13. Chang, P.C., Chen, S.H., Fan, C.Y.: Mining gene structures to inject artificial chromosomes for genetic algorithm in single machine scheduling problems. *Applied Soft Computing Journal* **8**(1), 767–777 (2008)
14. Chang, P.C., Chen, S.H., Fan, C.Y.: A hybrid electromagnetism-like algorithm for single machine scheduling problem. *Expert Systems with Applications* **36**(2, Part 1), 1259–1267 (2009)
15. Chang, P.C., Chen, S.H., Liu, C.H.: Sub-population genetic algorithm with mining gene structures for multiobjective flowshop scheduling problems. *Expert Systems with Applications* **33**(3), 762–771 (2007)
16. Chang, P.C., Chen, S.H., Mani, V.: A Hybrid Genetic Algorithm with Dominance Properties for Single Machine Scheduling with Dependent Penalties. *Applied Mathematical Modeling* **33**(1), 579–596 (2009)
17. Chang, P.C., Lee, H.C.: A greedy heuristic for bicriterion single machine scheduling problems. *Computers and Industrial Engineering* **22**(2), 121–131 (1992)
18. Chen, C.L., Vempati, V.S., Aljaber, N.: An application of genetic algorithms for flow shop problems. *European Journal of Operational Research* **80**(2), 389–396 (1995)
19. Chen, S.H., Chang, P.C., Chen, M.C., Chen, Y.M.: Dominance Properties for Self-Guided GA in Solving the Single Machine Scheduling Problems. *Proceedings of the CI-Sched 2009* **5227**, 292–299 (2008)
20. Chen, S.H., Chang, P.C., Zhang, Q.: Self-Guided genetic algorithm. *Lecture Notes in Artificial Intelligence* **5227**, 292–299 (2008)
21. Chen, S.H., Chen, M.C.: Convergency Analysis of Some Estimation of Distribution Algorithms in Detail. *Technical Report* (1), 1–7 (2009)
22. Corne, D., Dorigo, M., Glover, F., Dasgupta, D., Moscato, P., Poli, R., Price, K.: *New ideas in optimization*. McGraw-Hill Ltd., UK Maidenhead, UK, England (1999)
23. Dong, X., Huang, H., Chen, P.: An improved NEH-based heuristic for the permutation flowshop problem. *Computers and Operations Research* **35**(12), 3962–3968 (2008)
24. Framinan, J.M., Gupta, J.N.D., Leisten, R.: A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society* **55**(12), 1243–1255 (2004)
25. Gambardella, L., Taillard, E., Dorigo, M.: Ant colonies for the QAP. *Journal of the Operational Research Society* **50**(2), 167–176 (1999)
26. Garey, M., Johnson, D., Sethi, R.: The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research* **1**(2), 117–129 (1976)

27. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. *European Journal of Operational Research* **130**(3), 449–467 (2001)
28. Hansen, P., Mladenović, N., Perez-Britos, D.: Variable Neighborhood Decomposition Search. *Journal of Heuristics* **7**(4), 335–350 (2001)
29. Harik, G., Lobo, F., Goldberg, D.: The compact genetic algorithm. *Evolutionary Computation*, *IEEE Transactions on* **3**(4), 287–297 (1999)
30. Hejazi, S., Saghafian, S.: Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research* **43**(14), 2895–2929 (2005)
31. Hofmeyr, S., Forrest, S.: Architecture for an Artificial Immune System. *Evolutionary Computation* **8**(4), 443–473 (2000)
32. Hunt, J., Cooke, D.: Learning using an artificial immune system. *Journal of Network and Computer Applications* **19**(2), 189–212 (1996)
33. Iyer, S., Saxena, B.: Improved genetic algorithm for the permutation flowshop scheduling problem. *Computers and Operations Research* **31**(4), 593–606 (2004)
34. Jouglet, A., Savourey, D., Carlier, J., Baptiste, P.: Dominance-based heuristics for one-machine total cost scheduling problems. *European Journal of Operational Research* **184**(3), 879–899 (2008)
35. Lenstra, J., Kan, A., Brucker, P.: Complexity of machine scheduling problems. *Stud. integer Program., Proc. Workshop Bonn* (1975)
36. Li, G.: Single machine earliness and tardiness scheduling. *European Journal of Operational Research* **96**(3), 546–558 (1997)
37. Li, X.P., Wang, Y.X., Wu, C.: Heuristic algorithms for large flowshop scheduling problems. pp. 2999–3003 (2004)
38. Liaw, C.F.: A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. *Computers and Operations Research* **26**(7), 679–693 (1999)
39. Luo, X., Chu, C., Wang, C.: Some dominance properties for single-machine tardiness problems with sequence-dependent setup. *International Journal of Production Research* **44**(17), 3367–3378 (2006)
40. Luo, X., Chu, F.: A branch and bound algorithm of the single machine schedule with sequence dependent setup times for minimizing total tardiness. *Applied Mathematics and Computation* **183**(1), 575–588 (2006)
41. Miettinen, K.: *Nonlinear Multiobjective Optimization*. Springer (1999)
42. Montgomery, D.C.: *Design and Analysis of Experiments* (2001)
43. Muhlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions I. Binary parameters. *Lecture Notes in Computer Science* **1141**, 178–187 (1996)
44. Murata, T., Ishibuchi, H., Tanaka, H.: Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering* **30**(4), 1061–1071 (1996)
45. Nawaz, M., Ensco, E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA* **11**(1), 91–95 (1983)
46. Okabe, T., Jin, Y., Sendoff, B., Olhofer, M.: Voronoi-based estimation of distribution algorithm for multi-objective optimization. *Evolutionary Computation*, 2004. CEC2004. Congress on **2** (2004)
47. Osman, I., Wassan, N.: A reactive tabu search meta-heuristic for the vehicle routing problem with back-hauls. *Journal of Scheduling* **5**(4), 263–285 (2002)
48. Ow, P.S., Morton, T.E.: The Single Machine Early/Tardy Problem. *Management Science* **35**(2), 177–191 (1989)
49. Pasti, R., de Castro, L.: Bio-inspired and gradient-based algorithms to train MLPs: The influence of diversity. accepted by *Information Sciences* (2009)
50. Pelikan, M., Goldberg, D.E., Lobo, F.G.: A Survey of Optimization by Building and Using Probabilistic Models. *Computational Optimization and Applications* **21**(1), 5–20 (2002)
51. Peña, J., Robles, V., Larrañaga, P., Herves, V., Rosales, F., Perez, M.: GA-EDA: Hybrid Evolutionary Algorithm Using Genetic and Estimation of Distribution Algorithms. *Innovations In Applied Artificial Intelligence: 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2004, Ottawa, Canada, May 17-20, 2004: Proceedings* (2004)
52. Reeves, C.R.: A genetic algorithm for flowshop sequencing. *Computers and Operations Research* **22**(1), 5–13 (1995)
53. Reeves, C.R., Yamada, T.: Genetic Algorithms, Path Relinking, and the Flowshop Sequencing Problem. *Evolutionary Computation* **6**(1), 45–60 (1998)
54. Ruiz, R., Maroto, C.: A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research* **165**(2), 479–494 (2005)

55. Santana, R., Larrañaga, P., Lozano, J.: Combining variable neighborhood search and estimation of distribution algorithms in the protein side chain placement problem. *Journal of Heuristics* **14**, 519–547 (2008)
56. Sourd, F., Kedad-Sidhoum, S.: The One-Machine Problem with Earliness and Tardiness Penalties. *Journal of Scheduling* **6**(6), 533–549 (2003)
57. Sourd, F., Kedad-Sidhoum, S.: A faster branch-and-bound algorithm for the earliness-tardiness scheduling problem. *Journal of Scheduling* pp. 1–10 (2007)
58. Valente, J.M.S., Alves, R.A.F.S.: Improved heuristics for the early/tardy scheduling problem with no idle time. *Computers and Operations Research* **32**(3), 557–569 (2005)
59. Valente, J.M.S., Alves, R.A.F.S.: Heuristics for the early/tardy scheduling problem with release dates. *International Journal of Production Economics* **106**(1), 261–274 (2007)
60. Wang, X., Cheng, T.C.E.: A heuristic approach for tow-machine no-wait flowshop scheduling with due dates and class setups. *Computers and Operations Research* **33**(5), 1326–1344 (2006)
61. Wu, S.D., Storer, R.H., Chang, P.C.: One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers and Operations Research* **20**(1), 1–14 (1993)
62. Zhang, Q., Muhlenbein, H.: On the convergence of a class of estimation of distribution algorithms. *Evolutionary Computation, IEEE Transactions on* **8**(2), 127–136 (2004)
63. Zhang, Q., Sun, J., Tsang, E.: An Evolutionary Algorithm With Guided Mutation for the Maximum Clique Problem. *Evolutionary Computation, IEEE Transactions on* **9**(2), 192–200 (2005)
64. Zhang, Q., Sun, J., Tsang, E.: Combinations of estimation of distribution algorithms and other techniques. *International Journal of Automation and Computing* **4**(3), 273–280 (2007)
65. Zheng, D.Z., Wang, L.: An Effective Hybrid Heuristic for Flow Shop Scheduling. *The International Journal of Advanced Manufacturing Technology* **21**(1), 38–44 (2003)

Table 17 Selected results of Adaptive EA/G and EA/G-SGA employ different examined solutions in single machine scheduling problems

instance	Adaptive EA/G											
	50000			75000			100000			125000		
	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
skss422a	25656	25662.8	25712	25656	25659.7	25697	25656	25665.1	25697	25656	25669.4	25793
skss455a	6405	6420.1	6545	6405	6425.4	6545	6405	6428.4	6545	6405	6415.7	6545
skss488a	16862	16865.5	16888	16862	16862.9	16888	16862	16863.7	16888	16862	16863.7	16888
skss522a	29309	29326.2	29398	29309	29323.1	29398	29309	29326.7	29398	29309	29326.2	29398
skss555a	10187	10213.9	10256	10187	10224.2	10299	10187	10219.4	10301	10187	10223.3	10351
skss588a	24844	24847.9	24861	24844	24847.9	24861	24844	24845.8	24853	24844	24849.8	24870
skss622a	43048	43100.5	43371	43048	43078.7	43380	43048	43081	43273	43048	43078.6	43244
skss655a	16158	16163.2	16180	16158	16192.8	16616	16158	16208.1	16640	16158	16192.8	16640
skss688a	33551	33592	33686	33551	33624.4	33686	33551	33610	33686	33551	33580.7	33686
skss922a	88842	88872.1	88994	88841	88878.5	89100	88842	88871.6	89005	88841	88858.7	88956
skss955a	30582	30647	30804	30582	30674.8	31394	30582	30639.8	30800	30582	30628.4	30740
skss988a	81984	81986.4	81991	81984	81985.3	81989	81984	81985.2	81989	81984	81985.3	81989
EA/G-SGA												
instance	50000			75000			100000			125000		
	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max
skss422a	25656	25662.1	25704	25656	25661.9	25704	25656	25661.9	25704	25656	25661.9	25704
skss455a	6405	6424.8	6545	6405	6424.8	6545	6405	6424.8	6545	6405	6424.8	6545
skss488a	16862	16862.9	16888	16862	16862.9	16888	16862	16862.9	16888	16862	16862.9	16888
skss522a	29309	29315.5	29396	29309	29312.6	29396	29309	29312.5	29396	29309	29312.3	29396
skss555a	10187	10210.7	10299	10187	10210.5	10299	10187	10210.5	10299	10187	10209.9	10299
skss588a	24844	24844.3	24853	24844	24844.0	24844	24844	24844.0	24844	24844	24844.0	24844
skss622a	43048	43079.9	43244	43048	43079.9	43244	43048	43079.9	43244	43048	43079.9	43244
skss655a	16158	16235.2	16721	16158	16234.0	16721	16158	16234.0	16721	16158	16229.0	16640
skss688a	33551	33608.4	33665	33551	33608.4	33665	33551	33608.4	33665	33551	33608.4	33665
skss922a	88842	88876.4	89017	88842	88869.4	89005	88842	88869.4	89005	88842	88869.4	89005
skss955a	30582	30649.9	30795	30582	30642.7	30785	30582	30642.4	30785	30582	30641.9	30785
skss988a	81984	81989.4	82034	81984	81985.7	82033	81984	81985.6	82033	81984	81985.6	82033

Table 18 ANOVA result of the methods comparisons in makespan of Taillard flowshop scheduling problems

Source	DF	SS	Mean Square	F Value	P Value
instances	109	2.28E+11	2096295123	2.57E+06	<.0001
method	5	9763444.722	1952688.944	2392.83	<.0001
instances*method	545	27477265.03	50417.00005	61.78	<.0001
Error	19140	15619345.5	816.0577586		
Corrected Total	19799	2.29E+11			

Table 19 Duncan Post-hoc test of the permutation flowshop scheduling problems

Duncan Grouping	Mean	N	Method
A	5041.7076	3300	EA/G
B	5016.3655	3300	MGGA
C	5011.1815	3300	ACGA
C	5010.9885	3300	SGA
D	4985.8521	3300	NEH-EA/G
E	4972.5391	3300	NEH-ACGA

Table 20 Selected results of the permutation flowshop scheduling problems

E/A/G-SGA																			
Instance	SGA			MGGA			ACGA			E/A/G			NEH-ACGA			NEH-EA/G			
	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	
Ta051	3973	4011.8	4061	3940	3981.7	4072	3980	4016.4	4053	3997	4071.1	4134	3971	4002.2	4040	3972	4004.7	4054	
Ta056	3810	3872.9	3975	3857	3894.9	3972	3809	3870.8	3957	3842	3919.0	4054	3802	3829.8	3872	3818	3855.4	3892	
Ta060	3835	3925.2	4016	3862	3909.8	3992	3885	3927.9	3979	3911	3977.4	4073	3887	3939.2	3989	3918	3966.1	4009	
Ta061	5495	5501.5	5527	5495	5495.2	5498	5493	5506.5	5527	5493	5504.7	5527	5495	5503.6	5519	5495	5518.2	5519	
Ta066	5135	5142.7	5161	5137	5144.5	5150	5135	5148.2	5192	5137	5160.1	5248	5139	5139.0	5139	5139	5139.0	5139	
Ta070	5336	5353.1	5386	5342	5351.7	5372	5342	5353.6	5386	5342	5360.2	5389	5322	5338.9	5341	5341	5341.0	5341	
Ta071	5798	5894.1	6016	5863	5874.5	5942	5806	5895.2	5999	5818	5933.5	6057	5800	5821.8	5846	5846	5846.0	5846	
Ta076	5326	5378.4	5469	5344	5348.2	5389	5326	5388.9	5462	5342	5413.0	5508	5317	5328.3	5340	5317	5336.5	5344	
Ta080	5903	5914.3	6009	5903	5914.7	6004	5896	5920.6	6040	5903	5959.4	6101	5903	5903.0	5903	5903	5903.0	5903	
Ta081	6465	6561.5	6657	6487	6541.1	6665	6496	6564.5	6650	6495	6597.6	6683	6455	6488.3	6519	6454	6500.9	6526	
Ta086	6623	6718.6	6855	6548	6660.4	6751	6633	6701.1	6790	6645	6751.6	6885	6560	6616.8	6654	6606	6642.8	6650	
Ta090	6636	6742.0	6848	6684	6765.0	6877	6673	6764.4	6852	6736	6797.3	6897	6599	6621.8	6651	6603	6629.5	6654	
Ta091	10950	10993.5	11086	10940	10949.0	11032	10950	10982.1	11046	10953	11006.7	11079	10942	10942.0	10942	10942	10942.0	10942	
Ta096	10402	10488.3	10600	10492	10494.7	10533	10409	10485.8	10577	10475	10540.6	10658	10390	10423.4	10442	10401	10432.1	10439	
Ta100	10758	10840.5	10998	10743	10798.4	10909	10728	10826.8	10940	10803	10897.8	11004	10763	10779.9	10790	10776	10780.9	10794	
Ta101	11619	11722.9	11896	11739	11859.4	12005	11619	11717.0	11800	11623	11756.4	11884	11487	11520.9	11553	11486	11526.0	11560	
Ta106	11655	11764.8	11878	11811	11932.1	12086	11651	11770.0	11988	11604	11793.6	12061	11504	11544.6	11579	11519	11557.0	11601	
Ta110	11756	11907.0	12009	11878	12043.4	12172	11796	11908.1	12010	11803	11916.2	12072	11714	11781.1	11828	11753	11792.8	11836	

An Owner-centric Metric for the Evaluation of Online Job Schedules

Uwe Schwiegelshohn

Abstract The paper presents a metric that is well suited to evaluate online job schedules for massively parallel processors. We point out the disadvantages of commonly used metrics like makespan or machine utilization. We suggest to use the well known total weighted completion time metric with an appropriate selection of job weights and show that this metric exhibits many properties that are similar to the properties of the makespan objective. In addition, it also supports the evaluation of online schedules and allows extensions to consider additional job based system policies. Our performance evaluation of this metric is based on competitive analysis.

1 Background

The common notation for theoretical scheduling problems $(\alpha|\beta|\gamma)$ was introduced by Graham et al. [1] and describes the fields *machine environment* (α), *job characteristics* (β), and *objective function* (γ). The job characteristics define the feasibility of a schedule while the objective function discriminates the feasible schedules. The objective function is based upon a metric that allows an ordering of the feasible schedules.

In practice, it is not always possible to determine which one of two given feasible schedules is better as many real world problems are multi-criteria problems. Moreover, it is sometimes difficult to find a metric for some objectives, like, for instance, the requirement of a robust schedule that maintains the same or a similar good objective value if some job characteristics change slightly. Nevertheless, in order to design good scheduling algorithms, we need a metric that represents the practical objectives in a suitable fashion. However, after introducing such a metric, we must prevent that this metric becomes an objective for its own sake. Instead we should repeatedly check whether the metric still satisfies its original purpose. This is particularly true when new or modified scheduling problems with new machine models or new job characteristics appear. Then we should be careful before transferring an old metric to the new problem. But using an established metric often allows us to apply existing algorithms

Uwe Schwiegelshohn
Robotics Research Institute, TU Dortmund University
E-mail: uwe.schwiegelshohn@udo.edu

and analysis methods. Therefore, it may be beneficial to start with a well known metric to obtain first hints regarding the structure of good schedules.

In this paper, we exemplarily consider a simple but practically relevant job scheduling problem: Scheduling of compute jobs on a massively parallel compute system like a cluster. Such a cluster can usually be expressed with the help of the identical machine model P_m and typically executes its jobs in space sharing mode, that is, it exclusively allocates a certain number of processors (or nodes or cores) to a job. The compute jobs may exhibit different forms of parallelism, like rigid and malleable, see the definitions provided by Feitelson et al. [2]. Informally, a rigid job keeps the same machine allocation during its execution while a malleable job can increase or decrease its degree of parallelism depending on the availability of machines. However, the workload of a malleable job may be a function of the actual degree of parallelism. If the workload is invariant of the number of allocated processors, we talk about divisible load scheduling, see Bharadwaj et al. [3].

Typically, many users independently submit their jobs to the cluster of a compute center over time. Therefore, the compute system is not aware of any precedence relation between any two jobs even if there is one. Further, we have an online scenario as the compute system has no knowledge about future job submissions. Finally, the system usually does not know the processing time of the job. It is true that many scheduling systems require users to provide runtime estimates for their jobs. But these estimates are supposed to protect resources (and users) from wasting resource time by faulty jobs as a job is typically aborted once it reaches its estimate. Nevertheless, some scheduling approaches like backfilling, see Lifka [4], use these estimates to generate better schedules. However, Lee et al. [5] have shown that these estimates are often very inaccurate effectively producing scheduling problems with some nonclairvoyant character.

The evaluation of a schedule requires a metric. This metric must satisfy two main conditions:

1. It must appropriately represent the goals of the resource owner and/or the users.
2. It must allow a quantitative evaluation of the schedule.

In the job scheduling scenario, users are interested to receive the (correct) results of jobs as early as possible and to pay as little as possible for the (reliable) execution of their jobs. Note that the exact relationship between response time and cost may not be a linear function. For instance, a user may be eager to receive the result before she leaves work in the evening but if this is not possible, it is sufficient that the result is available when she arrives next morning. Moreover, the time and the price criteria are likely to interfere with each other generating a bicriteria optimization problem that requires some form of compromise. The problem of defining an appropriate user-centric metric has been subject of previous research, see Lee and Snaveley [6]. On the other hand, resource owners want to receive the largest possible return for providing their resources. To this end, they may assign users to different user groups and give different priorities to these groups according to the agreed price in case of any resource shortage.

In this paper, we do not consider the user's but the owner's point of view and start with the assumption that there are no user priorities, that is, the price for a unit of consumed processor resources is fixed. Hence, the owner is mainly interested in a high system utilization. The utilization of a machine within a given time interval is the ratio of the total compute time used by all jobs to the total available compute time. But utilization is traditionally not the method of choice in theoretical job scheduling problems. Instead, the makespan criterion $C_{\max}$ is commonly selected. As the makespan

of a schedule is the last completion time of any job in this schedule there is a close relationship to utilization: x percent reduction in the makespan corresponds to $\frac{x}{100\%-x}$ percent increase in utilization. But note that we must compare different intervals to determine this change.

In addition to the above mentioned major properties, a suitable metric often also possesses secondary characteristics:

1. It allows an easy evaluation.
2. It supports extensions to more complex criteria defined by the participants within the scenario.

An easy evaluation is particularly important in online scheduling problems where the schedule does not really end. For instance, if the scheduling system uses a rule based approach, see, for instance, Franke et al. [7], it must continuously evaluate the schedule in order to determine when to switch to another set of rules. In this case, the metric should reliably describe the most recent part of the schedule. Further, once a suitable solution to the scheduling system has been obtained, participants (owner and users) often tend to look for an even better approximation of their actual goals by incorporating secondary objectives. A suitable metric would allow such an extension without requiring a completely new design process.

Unfortunately, the secondary goals are not met by the makespan objective for online schedules of parallel machines. For instance, the last job may have a significant influence on the quality of the schedule. Let us assume that a schedule achieves 100% utilization for the last k jobs, that is, the schedule has a minimal makespan and optimal load balance. However, the next job immediately changes this picture: Particularly if this next job has a sufficiently large processing time the resulting makespan may be as bad as a factor of almost 2 away from the optimum for systems with many processors, see Graham [8] and Fig. 1. Note that in Fig. 1, as in all graphical representations of schedules in this paper, the vertical and horizontal axis denote time and machines, respectively. To support a better intuitive understanding of the figures, we represent a rigid parallel job as single rectangle although the processors need not be allocated in a contiguous fashion. Moreover as the makespan metric is based on the completion time of one job only, it cannot be simply extended to address user groups.

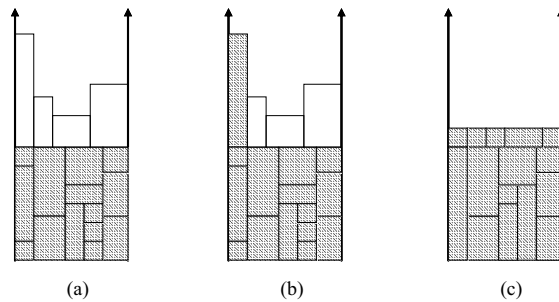


Fig. 1 Online schedules with k jobs (a) and with $k + 1$ jobs (b); optimal makespan schedule with $k + 1$ jobs (c)

Alternatively, we can directly use utilization as our objective. In order to avoid the first problem with the makespan metric, we may select an interval that ends at the

actual time. This is a good approach if we have no information about the processing time of actually running jobs (nonclairvoyant scheduling). However, it does not allow the consideration of processing time estimates. Further, it is not possible to extend the utilization metric in a simple fashion to distinguish the jobs of different user groups.

In this paper, we present a scheduling objective that is well suited to evaluate online job schedules of massively parallel processors. We start with another objective frequently applied in theoretical scheduling problems: *Total completion time*. It is the sum of the completion times of all jobs in a schedule S . This objective considers each individual job separately but is biased towards some jobs: Fig. 2 shows that the left schedule has a significant lower total completion time than the right schedule although the utilization in both schedules is the same. Clearly, the total completion time objective favors sequential jobs over parallel jobs if they have the same processing time. Therefore, a user will benefit if he can split his parallel job into many sequential jobs although he requires the same total computation time in both cases. The same is true if he splits a long running job into many short running jobs (with precedence constraints).



Fig. 2 Total completion time of 2 schedules with a parallel und several sequential jobs

However, we are interested in a metric that does not automatically provide a preferential treatment to short or long, parallel or sequential jobs. In our view, a basic owner-centric objective should primarily be neutral regarding job characteristics. In addition, the objective should enable extensions to consider the policy of the specific system. Therefore, we use the *total weighted completion time* metric with the resource consumption (processing time times parallelism for rigid jobs) of a job being its weight. This metric guarantees the desired neutrality. Then it is up to the system owner to assure preference of certain jobs by modifying the weights accordingly. This modification may be based on any job characteristic.

There are only very few results regarding the performance of simple algorithms with respect to this metric. For sequential jobs, this weight selection is similar to the l_2 -norm that has already been subject of several studies, see Bansal and Pruhs [9] or Avidor et al. [10]. Schwiegelshohn and Yahyapour [11] have also used this metric to guarantee fairness in preemptive scheduling of rigid parallel jobs. In this paper, we evaluate this metric for nonclairvoyant sequential and rigid parallel jobs on a machine with parallel identical processors (P_m -model). To this end, we use the competitive analysis, that is, we determine for all input sets the maximal deviation of a generated schedule from the corresponding optimal schedule with respect to this metric. The competitive factor is the largest ratio between these two values, see Section 2. We compare this competitive factor with the corresponding value obtained for the conventional makespan objective.

2 Definitions and Previous Results

In our basic scenario, we use the P_m machine environment, that is, all m processors of a parallel machine are identical. Occasionally, we also refer to the single machine environment ($\alpha = 1$) as a reference. Our jobs are either sequential or rigid ($\beta = size_j$). A rigid parallel job J_j requires the concurrent availability of $size_j$ processors during its whole execution. These $size_j$ processors must be exclusively allocated to job J_j . Further, jobs are submitted over time ($\beta = r_j$), that is, job J_j is unknown before its release date r_j . Again as a reference, we also discuss problems with all jobs being released at time 0. Contrary to other online problems, we do not demand that all $size_j$ processors are assigned to a job J_j at its submission date. Instead the processor-job allocation is delayed until the required number of processors is actually available. Moreover, we assume that the processing time p_j of job J_j becomes only known once the job has finished its execution, that is, we consider nonclairvoyant scheduling ($\beta = nc$). If all jobs are sequential or released at time 0 the problem notation does not use the constraints $\beta = size_j$ or $\beta = r_j$, respectively.

$C_j(S)$ denotes the completion time of job J_j in schedule S . If the schedule is nonambiguous then we omit S . Our primary schedule objective is the total weighted completion time with the weight of job J_j being its resource consumption $p_j \cdot size_j$: $\gamma = \sum p_j size_j C_j$. As reference problems we use corresponding problems with the makespan objective $C_{\max} = \max_j \{C_j\}$ that have extensively been discussed in earlier publications.

For a given scheduling algorithm, we determine upper bounds for the ratio $\frac{\sum_j p_j size_j C_j(S)}{\sum_j p_j size_j C_j(OPT)}$ with S and OPT being the schedules produced by the algorithm and the optimal schedule, respectively. This bound is called an approximation factor for deterministic scheduling problems and a competitive factor for online problems. Note that this is a worst case evaluation. An algorithm may perform much better in case of a real workload. Nevertheless, a worst case analysis may provide interesting information about possible bad situations.

For the total weighted completion time problem on a single processor $1 || \sum w_j C_j$ without release dates, a nondelay scheduling in descending order of the Smith ratio $\frac{w_j}{p_j}$ generates an optimal schedule. We say that a schedule is nondelay, if no processor is kept idle while a job is waiting, see Pinedo [12]. Therefore, any nondelay schedule is optimal for the $1|ncv| \sum p_j C_j$ problem. This also holds for the corresponding online problem $1|r_j, ncv| \sum p_j C_j$ while the problem with arbitrary weights ($1|r_j| \sum w_j C_j$) is NP-hard in the strong sense. Concluding, we can state that the objective $\sum p_j C_j$ produces the same results as the makespan objective for single processors. Remember that although the processing time is part of the objective, its knowledge is not required by the algorithm. In the next section, we address machines with parallel identical processors.

3 Parallel Machines and Sequential Jobs

The problem $P_m || C_{\max}$ is strongly NP-hard for $m \geq 3$. List scheduling with an arbitrary list order guarantees the approximation factor $2 - \frac{1}{m}$, see Graham [8]. This approximation factor is tight for nonclairvoyant scheduling. This result also holds for offline and online problems with rigid parallel jobs, see Graham [13] and Naroska

and Schwegelshohn [14], respectively. The problem $P_m || \sum p_j C_j$ is NP-hard as partition reduces to $P2 || \sum p_j C_j$. List scheduling with an arbitrary list order guarantees a tight approximation factor of $\frac{1+\sqrt{2}}{2}$ for this problem, see Kawaguchi and Kyan [15]. Schwegelshohn and Yahyapour [11] showed a competitive factor of 2 for the problem $P_m | r_j, size_j, ncv | \sum p_j C_j$ if $size_j \leq \frac{m}{2}$ holds for all jobs. While this demonstrates the existence of a constant competitive factor for the problem $P_m | r_j, ncv | \sum p_j C_j$, the gap between the upper bound 2 and the lower bound $\frac{1+\sqrt{2}}{2}$ is still rather large. In this section, we close this gap significantly.

Theorem 1 *List scheduling has a competitive factor of at most 1.25 for the problem $P_m | r_j, ncv | \sum p_j C_j$.*

Before presenting the proof for this problem, we use a lemma that has already been introduced as Corollary 6 in a publication by Schwegelshohn and Yahyapour [11]:

Lemma 1 *Let S and OPT be a legal schedule and the optimal schedule of an arbitrary problem instance $P_m | r_j | \sum p_j C_j$ with n jobs, respectively. We split an arbitrary job J_k into two jobs J_{n+1} and J_{n+2} such that $0 < p_{n+1} < p_k$, $p_{n+2} = p_k - p_{n+1}$, $r_{n+1} = r_k$, and $r_{n+2} = r_k + p_{n+1}$ hold. Schedule S' is derived from schedule S by replacing the execution of job J_k with the execution of job J_{n+1} immediately followed by the execution of job J_{n+2} .*

$$\text{This results in } \frac{\sum_j p_j C_j(S')}{\sum_j p_j C_j(OPT')} \geq \frac{\sum_j p_j C_j(S)}{\sum_j p_j C_j(OPT)}.$$

Although Lemma 1 always produces legal schedules the resulting schedule need not be nondelay, see the example of Fig. 3 that describes a system with all original jobs being released at time 0. There, splitting of the long job does not yield a nondelay schedule.

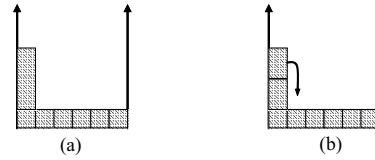


Fig. 3 Nondelay schedule (a) and schedule after job splitting (b)

We can use Lemma 1 to transform an arbitrary schedule into a *target schedule*: We split every job J_j if $p_j > \epsilon$ and the splitting does not lead to a nondelay schedule. Due to Lemma 1, it is sufficient to consider this type of schedule to determine the competitive factor of nondelay schedules. Finally, we reduce the release dates of each job without changing the schedule until any further reduction would violate the nondelay property of this schedule. As this reduction cannot increase the total completion time of the optimal schedule the ratio $\frac{\sum_j p_j C_j(S)}{\sum_j p_j C_j(OPT)}$ cannot decrease. Now, a target schedule contains short jobs ($p_j \leq \epsilon$) and long jobs ($p_j > \epsilon$), and has four main properties:

1. The schedule is nondelay.

2. At least one machine is idle at the starting time of every long job.
3. For every long job J_j and every positive value $r_j < t < r_j + p_j$, there is at least one processor idle at some time instance of the interval $[t, t + C_j(S) - r_j - p_j]$.
4. All jobs that start in an interval (including the boundaries) without any idle processor have the same release date.

If the second property does not hold then we can split a long job at time $C_j(S) - p_j + \epsilon$. If the third property does not hold then we can split a long job J_j at time $t + C_j(S) - r_j - p_j$, see Fig. 4. If the fourth property does not hold then we can further reduce the release date of a job.

For any target schedule S , we consider a nondelay schedule OPT that starts all long jobs at their release dates. Due to the properties of target schedules, such a schedule always exists: The first property of schedule S guarantees that for a long job J_i , all processors are busy in the interval $[r_i, C_i(S) - p_i]$. Assume that job J_i and a long job J_j with $r_j < r_i$ are executed on the same processor in schedule S . Then the inequality $r_j + p_j > r_i$ violates the third property of target schedules, see Fig. 5. Therefore, no long job J_j that starts at r_j can prevent the next long job on this processor to start at its release date as well. For reasons of convexity, schedule OPT has the minimal total completion time if all short jobs have the same processing time. This can be achieved with an arbitrarily small deviation if ϵ is small enough.

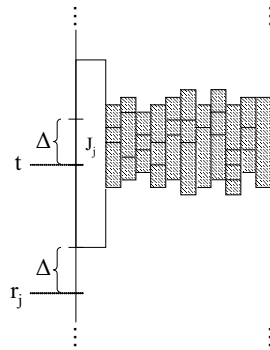


Fig. 4 Splitting of a long job in a busy interval

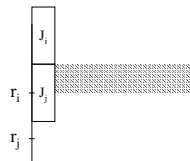


Fig. 5 Rearranging long jobs in a schedule

Now we describe the proof of Theorem 1:

Proof We use induction on the number of different release dates. Problem $P_m || \sum p_j C_j$ is the basis of the proof as $\frac{1+\sqrt{2}}{2} < 1.25$ holds. As hypothesis we assume that Theorem 1 is correct for all problem instances with at most i different release dates. Next we consider an arbitrary problem instance with $i+1$ different release dates and a nondelay target schedule S . Let r be the maximum release date and $\Theta = \{J_j | r_j = r\}$ be the set of jobs that are released at time r .

Due to the fourth property of target schedules, no job J_j with $r_j < r$ starts at time r or later. Let $t_s(S)$ be the last start time of any job in schedule S .

The processing time of any job J_j with $r_j < r$, $p_j > r - r_j$, and is reduced to $r - r_j$. This reduction does not decrease the completion time of any job $J_i \in \Theta$ in schedule S while it cannot increase the contribution of jobs in Θ to the total completion time of schedule OPT . Although the ratio $\frac{\sum_j p_j C_j(S)}{\sum_j p_j C_j(OPT)}$ may increase or decrease due to this reduction, the total contribution of all jobs with a smaller release date than r remains bounded by our hypothesis.

Now assume a job J_j with $r_j < r$, $p_j > r - r_j$, and $C_j(S) - t_s(S) < p_j + r_j - r$. We split this job into one long job with release date r_j and processing time $r - r_j$ and one (long) job with release date r and processing time $p_j - r + r_j$. Note that in schedule S , this splitting occurs in the interval $[r, t_s(S))$ and does not violate the nondelay property. However, we may have to do some further splitting in order to preserve the third property of target schedules among the jobs of Θ . Therefore, no long job J_j with $r_j < r$ can complete after time r in schedule OPT .

If some short jobs complete after time r in schedule OPT although they have been submitted before r then we simply split a long job J_i with $r_i < r$ and $r < C_i(S)$ at some time in the interval $[r, t_s(S)]$ such that no job J_j completes after time r in schedule OPT . Observe that this splitting reduces the total processor-time product of all jobs submitted before r . To exactly achieve the desired result, we may have to repeat this splitting or to (repeatedly) duplicate the number of machines and all jobs. The latter procedure does not change the ratio $\frac{\sum_j p_j C_j(S)}{\sum_j p_j C_j(OPT)}$.

With W_r being the total processor-time product of all jobs with a release date smaller than r in the interval $[r, \infty)$ of schedule S we assume that these jobs completely occupy the interval $[r, r + \frac{W_r}{m})$, see Fig. 6. This can only increase the total completion time contribution of jobs $J_i \in \Theta$ in schedule S . Now, we use the utilization bound $\frac{4}{3}$ of Hussein and Schwiegelshohn [16]. This leads to $W_r \leq \frac{1}{4}m \cdot r$.

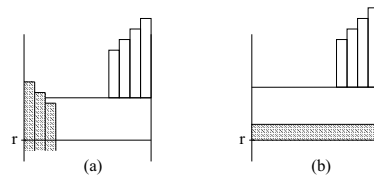


Fig. 6 Schedule S (a) and assumed processor occupation (b)

Finally, we can combine the total completion times to obtain the desired result:

$$\begin{aligned}
 \frac{\sum_j p_j C_j(S)}{\sum_j p_j C_j(OPT)} &= \frac{\sum_{J_j \in \Theta} p_j C_j(S) + \sum_{J_j \notin \Theta} p_j C_j(S)}{\sum_{J_j \in \Theta} p_j C_j(OPT) + \sum_{J_j \notin \Theta} p_j C_j(OPT)} \\
 &< \frac{\sum_{J_j \in \Theta} p_j (C_j(S) - 1.25r) + 1.25r \sum_{J_j \in \Theta} p_j + 1.25 \sum_{J_j \notin \Theta} p_j C_j(OPT)}{\sum_{J_j \in \Theta} p_j (C_j(OPT) - r) + r \sum_{J_j \in \Theta} p_j + \sum_{J_j \notin \Theta} p_j C_j(OPT)} \\
 &\leq \frac{\frac{1+\sqrt{2}}{2} \sum_{J_j \in \Theta} p_j (C_j(OPT) - r) + 1.25r \sum_{J_j \in \Theta} p_j + 1.25 \sum_{J_j \notin \Theta} p_j C_j(OPT)}{\sum_{J_j \in \Theta} p_j (C_j(OPT) - r) + r \sum_{J_j \in \Theta} p_j + \sum_{J_j \notin \Theta} p_j C_j(OPT)} \\
 &< 1.25
 \end{aligned}$$

The determined competitive factor for the problem $P_m|r_j, ncv| \sum p_j C_j$ is 1.25 while $\frac{1+\sqrt{2}}{2} \approx 1.207$ is a lower bound. Therefore, the gap is small (< 0.05).

4 Parallel Machines and Parallel Jobs

In this section, we address rigid parallel jobs that require a fixed number of processors during all of their processing time. We use the notation $P_m|r_j, size_j, ncv| \sum p_j size_j C_j$ to describe the corresponding scheduling problem. Unfortunately, List scheduling with an arbitrary order does not perform very well for this problem contrary to the problem $P_m|size_j, ncv|C_{\max}$ that guarantees the tight competitive factor $2 - \frac{1}{m}$, see Garey [13]. A simple example using only two jobs J_1 and J_2 with $p_1 = \sqrt{m}$, $p_2 = 1$, $size_1 = 1$, and $size_2 = m$ shows that the competitive factor of List scheduling may be as bad as $\frac{m\sqrt{m}+2m}{2m+\sqrt{m}}$, see Fig. 7. As both jobs cannot be executed concurrently, both schedules have the same optimal makespan. But in an online scenario it is better to schedule the parallel job first as this order offers options for the scheduling of additional jobs (left schedule). In the right schedule, any idle processor in the interval $[0, \sqrt{m})$ can only be utilized by the backfilling method [4]. But backfilling requires at least an estimate of the processing times. Therefore, the new objective allows to distinguish between these two schedules in an intuitive fashion.

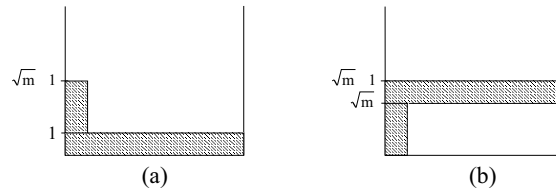


Fig. 7 Optimal schedule (a) and bad List schedule (b)

However, we can guarantee acceptable solutions for the problem $P_m|size_j, ncv| \sum p_j size_j C_j$ by sorting the jobs in descending order of their degree of parallelism ($size_j$) before applying List scheduling.

Theorem 2 List scheduling in descending order of the $size_j$ value has an approximation factor of 2 for the problem $P_m|size_j, ncv| \sum p_j size_j C_j$.

Again we use a well known lemma, see Queyranne [17], before presenting the proof for Theorem 2.

Lemma 2 Let S be a nondelay schedule for the problem $1|ncv| \sum p_j C_j$. Then we have

$$\sum_j p_j C_j(S) = \frac{1}{2} \left(\left(\sum_j p_j \right)^2 + \sum_j p_j^2 \right).$$

Lemma 2 confirms our previous statement that the objective value of nondelay schedules for the problem $1|ncv| \sum p_j C_j$ is independent of the scheduling order. Now, we return to the proof of Theorem 2.

Proof Let $t_s(S)$ be the first time instance at which at most $\frac{m}{2}$ processors are used. Then no job can start later than $t_s(S)$. Let Θ be the set of all jobs that start at time $t_s(S)$, and let job J_i be the job with the smallest $size_j$ value among all jobs in Θ . Due to List scheduling, there are less than $size_i$ processors idle at any time instance in the interval $[0, t_s(S))$. Next we use several steps to transform schedule S ((a) in Fig.8).

1. We split every job $J_j \notin \Theta$ into $size_j$ sequential jobs with the same processing time ((b) in Fig.8). Note that the resulting schedule is not a nondelay schedule anymore. This transformation does not change the objective value of schedule S while the corresponding value of the optimal schedule cannot increase.
2. We apply Lemma 1 until no job that completes before $t_s(S)$ has a larger processing time than ϵ ((c) in Fig.8). This may add further jobs to Θ and transforms schedule S into schedule S'' .
3. We combine all jobs in Θ into a single job J_k with $p_k = \frac{\sum_{J_j \in \Theta} p_j^2}{\sum_{J_j \in \Theta} p_j}$ and $size_k = \frac{(\sum_{J_j \in \Theta} p_j)^2}{\sum_{J_j \in \Theta} p_j^2}$ ((d) in Fig.8). Note that $size_k \geq size_i$ holds.

Further, we rearrange schedule S'' such that exactly $m - size_k$ processors are used at any time instance before job J_k starts. This rearrangement produces schedule S' and results in $t_s(S') \geq t_s(S)$ ((e) in Fig.8). Therefore, the objective value of schedule S' cannot decrease while the objective value of the optimal schedule after the transformations ((f) in Fig.8) cannot increase due to reasons of convexity, see Lemma 2.

Finally, we must determine the worst case values for p_k and $size_k$ as functions of m and $t_s(S')$. Using Lemma 2, this leads to a rather simple numerical optimization problem and produces the approximation factor 2 for $\epsilon \rightarrow 0$, $p_k = 0$, and $size_k = \frac{m}{2}$.

The following example shows that the approximation factor is tight for this algorithm.

Example 1 Assume a problem instance with $2k + 2$ jobs and an even number of processors m :

- Job J_1 with $p_1 = 1$ and $size_1 = m - 1$;

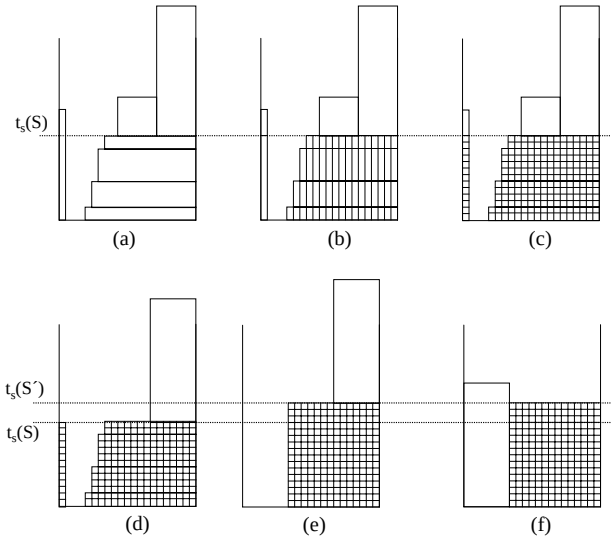


Fig. 8 Transformation steps of the proof of Theorem 2

- Jobs J_2 to J_{2k+1} with $p_j = 1$ and $size_j = \frac{m}{2}$;
- Job J_{2k+2} with $p_{2k+2} = 2k$.

List scheduling produces schedule S and starts jobs J_1 and J_{2k+2} concurrently at time 0. Afterwards jobs J_2 to J_{2k+1} are scheduled one after the other in a nondelay fashion, see the left schedule in Fig. 9. The optimal schedule OPT will schedule jobs J_2 to J_{2k+1} in pairs first and start jobs J_1 and J_{2k+2} at time k , see the right schedule in Fig. 9. With the help of Lemma 2, this leads to

$$\lim_{m,k \rightarrow \infty} \frac{\sum_j p_j C_j(S)}{\sum_j p_j C_j(OPT)} = 2$$

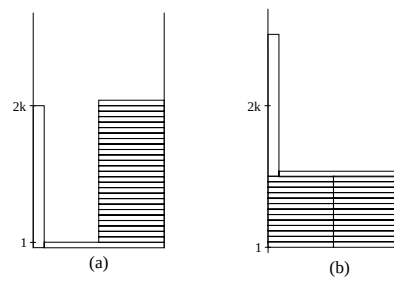


Fig. 9 Worst case example for List scheduling in descending order of the $size_j$ parameter

Unfortunately, List scheduling in descending order of the $size_j$ value is not applicable for the problem $P_m|r_j, size_j, ncv| \sum p_j size_j C_j$ as the nondelay property may lead to a preferred scheduling of sequential jobs and therefore highly parallel jobs may be subject to *starvation* despite the order of the list. Moreover, there is no algorithm with a constant competitive factor as demonstrated by Theorem 3.

Theorem 3 *The lower bound of the competitive factor for the problem $P_m|r_j, size_j, ncv| \sum p_j size_j C_j$ approaches $\frac{m\sqrt{m+2m}}{2m+\sqrt{m}}$.*

Proof First, the adversary submits a single job J_1 with $size_1 = 1$ at time 0. Eventually, the scheduler must schedule this job to prevent an arbitrarily bad competitive factor. Then the adversary immediately submits a parallel job J_2 with $size_2 = m$. Finally, he selects the processing times of both jobs to match the example of Fig. 7 while considering the submission time r_2 .

In practice however, this result is not really disturbing as there are only few highly parallel jobs in real workloads. If each parallel job uses at most 50% of the processors of a machine then simple List schedule guarantees already a competitive factor of 2, see Schwiegelshohn and Yahyapour [11]. This competitive factor decreases further if the bound on the parallelism of the individual jobs becomes tighter. The new metric has the advantage that we can incorporate preferences of user groups or special jobs by simply multiplying the weight of these jobs with an owner defined constant. Further, it is easy to consider only recently submitted jobs in an online schedule. This enables an online evaluation of the schedule and therefore adaptation of the scheduling strategy.

References

1. Graham, R., Lawler, E., Lenstra, J., Kan, A.R.: Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* **15**, 287–326 (1979)
2. Feitelson, D., Rudolph, L., Schwiegelshohn, U., Sevcik, K., Wong, P.: Theory and practice in parallel job scheduling. In: D. Feitelson, L. Rudolph (eds.) *IPPS'97 Workshop: Job Scheduling Strategies for Parallel Processing*, pp. 1–34. Springer-Verlag, Lecture Notes in Computer Science LNCS 1291 (1997)
3. Bharadwaj, V., Robertazzi, T.G., Ghose, D.: *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA (1996)
4. Lifka, D.: The ANL/IBM SP Scheduling System. In: D. Feitelson, L. Rudolph (eds.) *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing*, pp. 295–303. Springer-Verlag, Lecture Notes in Computer Science LNCS 949 (1995)
5. Lee, C.B., Schwartzman, Y., Hardy, J., Snavely, A.: Are user runtime estimates inherently inaccurate? In: *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 153–161 (2004)
6. Lee, C.B., Snavely, A.E.: Precise and realistic utility functions for user-centric performance analysis of schedulers. In: *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, pp. 107–116 (2007)
7. Franke, C., Hoffmann, F., Lepping, J., Schwiegelshohn, U.: Development of scheduling strategies with genetic fuzzy systems. *Applied Soft Computing Journal* **8**(1), 706–721 (2008)
8. Graham, R.L.: Bounds for certain multiprocessor anomalies. *Bell System Technical Journal* **45**, 1563–1581 (1966)
9. Bansal, N., Pruhs, K.: Server scheduling in the l_p norm: a rising tide lifts all boats. In: *ACM Symposium on Theory of Computing (STOC)*, pp. 242–250 (2003)
10. Avidor, A., Azar, Y., Sgall, J.: Ancient and new algorithms for load balancing in the p norm. *Algorithmica* **29**(3), 422–441 (2001)

11. Schwiegelshohn, U., Yahyapour, R.: Fairness in parallel job scheduling. *Journal of Scheduling* **3**(5), 297–320 (2000)
12. Pinedo, M.: *Scheduling: Theory, Algorithms, and Systems*, second edn. Prentice-Hall, New Jersey (2002)
13. Graham, R.: Bounds on multiprocessor timing anomalies. *SIAM Journal of Applied Mathematics* **17**, 416–429 (1969)
14. Naroska, E., Schwiegelshohn, U.: On an online scheduling problem for parallel jobs. *Information Processing Letters* **81**(6), 297–304 (2002)
15. Kawaguchi, T., Kyan, S.: Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM Journal on Computing* **15**(4), 1119–1129 (1986)
16. Hussein, M., Schwiegelshohn, U.: Utilization of nonclairvoyant online schedules. *Theoretical Computer Science* **362**, 238–247 (2006)
17. Queyranne, M.: Structure of a simple scheduling polyhedron. *Mathematical Programming* **58**(2), 263–285 (1993)

A deterministic resource scheduling model in epidemic logistics

Costas P. Pappis · Nikolaos P. Rachaniotis · Tom Dasaklis

Abstract

Available resources to tackle an epidemic infection are usually limited, while time and effort required to control it are increasing functions of the starting time of the containment effort. The problem of scheduling limited available resources, when there are several areas where the population is infected, is considered. A deterministic model, appropriate for large populations, where random interactions can be averaged out, is used for the epidemic's rate of spread. The problem is tackled using the concept of deteriorating jobs, i.e. the model represents increasing loss rate as more susceptibles become infected, and increasing time and effort needed for the epidemic's containment.

Keywords: Scheduling, Disaster and Crisis Management, Deteriorating jobs

1. Introduction

A serious epidemic is a problem that tests the ability of a nation to effectively protect its population, to reduce human loss and to rapidly recover. Very often such a problem may acquire worldwide dimensions. According to the World Health Organization, the world is facing an infectious disease crisis of global proportions, which is responsible for more than 13 million deaths and multimillion infections per year (for example more than 40 million people worldwide are infected with human immunodeficiency virus (HIV), over 300 million cases of infection with sexually transmitted diseases other than HIV occur worldwide each year and about 8 million people become sick with tuberculosis each year of which 2 million people die ([3])). Diarrheal diseases, tuberculosis, malaria, measles, influenza and pneumonia also cause significant numbers of direct infectious disease deaths. In addition there are several infectious

Dept. of Industrial Management, University of Piraeus, 80 Karaoli & Dimitriou Str.,
18534 Piraeus, Greece, tel. +30 210 4142150
e-mails: pappis@unipi.gr, nraxan@unipi.gr, dasaklis@unipi.gr

diseases that can cause death indirectly (for example, human papilloma virus, schistosomiasis, Hepatitis B and C).

Epidemiology is the study of the distribution and determinants of chronic and infectious disease prevalence in humans. By creating mathematical models, diseases can be studied and methodologies can be applied in order to handle them and save human lives by implementing these methodologies when an epidemic appears. This kind of modeling grew rapidly after World War II, when research on the spread of epidemics became a high priority.

Although infectious diseases pose a serious threat to public health, resources for controlling them are often limited. Decision makers must determine how to allocate and schedule limited resources among subpopulations and controlling the spread of infectious diseases is a challenging public health problem. Additionally, characteristics related to the spread of an epidemic may differ across sub-populations. Consequently, the problem's uniqueness and severe impacts demand dynamic, real-time and effective solutions, thus making the topic very suitable for OR/MS research in the area of Disaster Operations.

Disaster Operations in general are considered to be a set of activities that are performed before, during and after a disaster with the goal of preventing loss of human life, reducing its impact on the economy and returning to a state of normalcy. The International Federation of Red Cross and Red Crescent Societies (IFRC) lists disaster types as: Hurricanes, cyclones and typhoons, floods, drought, earthquakes, volcanic eruption, epidemics, famine and food insecurity, man-made disasters, population movement and technological disasters. The reader is referred to the IFRC web site for the definitions of these disasters ([14]). An emergency response is one to catastrophic and disaster events and does not consider daily response of ambulance, police or fire departments to emergency calls (routine or every day emergencies). Emergency response efforts consist of two general stages: pre-event and post-event response. Pre-event tasks include predicting and analyzing potential dangers and developing necessary action plans for mitigation. Post-event response starts while the disaster is still in progress and at this stage the objective is to locate, allocate, coordinate and manage available resources.

In ([2]) it seems that there is no contribution from the scientific area of OR/MS to epidemics control in pure OR/MS literature. However, a concise survey can be found in ([4]). In the problems tackled in this area, a "production function" is utilized, which describes the relationship between the amount of money invested in interventions and the outcome. Usually it is assumed that there is a linear relationship (linear production function) between investment in a prevention program and its results. The two most commonly used objectives are to maximize the number of Quality Adjusted Life Years (QALY) gained and/or maximize the number of infections averted (or, equivalently, minimize the cumulative number of new infections). The techniques that have been used for resources' allocation in epidemics' control

are linear and integer programming models, numerical analysis procedures, cost-effectiveness analysis, simulation, non-linear optimization and control theory techniques. The problem has also been formulated as a knapsack one ([3], [4]) and heuristic algorithms are presented for solving the general problem.

Job scheduling research has been recently extended to situations with deteriorating (increasing) processing times, depending on the jobs' processing starting times. This situation appears in many real applications, e.g. scheduling industrial processors whose productivity reduces over time, scheduling industrial maintenance, repair and cleaning assignments, scheduling steel rolling mills and metallurgical processes, repayment of multiple loans, etc ([1], [5]). A new version of the problem is to find the optimal scheduling policy if the processing times deteriorate, i.e., increase depending on the processing starting time, and the jobs' values also deteriorate, i.e. decrease over time where the scheduling criterion is the maximization of the total jobs' remaining value at the time that processing of all jobs is completed ([13]).

The general machine scheduling problem has been used to solve the resource allocation problem after a severe earthquake ([7]), using a Simulated Annealing and a Tabu Search heuristic for calculating a resource schedule to minimize the total expected number of fatalities. The concept of deteriorating jobs' scheduling has been applied in the area of wildfires' suppression and scheduling in area contaminated with chemical or radioactive materials. More specifically:

In the area of wildfire suppression, there are two applications: a) In ([13]), the problem of scheduling a single fire fighting resource when there are several forest fires to be suppressed is considered. The time required to suppress a forest fire increases if there is a delay in the beginning of the containment's effort, while the value (monetary or of some other kind) of the affected area is assumed to decrease quadratically with the duration of the containment effort. Consequently, scheduling the available fire fighting resource in the case of multiple forest fires may be treated as a job-scheduling problem with deteriorating processing times, with the objective of maximizing the total remaining value of the burnt areas after the completion of the containment operation. b) In ([11]) the extension of the previously described problem of scheduling $n > 1$ jobs (forest fires) on $m > 1$ non-identical, parallel processors (fire fighting resources) was examined. The above problem with multiprocessor jobs (i.e. jobs that require more than one processor at the same moment of time) was stated and a model was formulated, the criterion being to maximize the total remaining value of the burnt areas. A real time synchronous heuristic algorithm was proposed, which was implemented and its efficiency was compared to real data (provided by Greek Fire Corps from a severe wildfire near Athens in June-July, 2007).

In the field of scheduling in a contaminated area with radio-active or chemical materials, ([8], [9]) examine the case of scheduling n jobs (e.g. rescue, de-activation or cleaning works) to be executed by a single worker. Precedence relations can be given on the set of jobs and preemption can be allowed. Each job work period should be followed by a rest period whose length depends on the start time and length of the work period. The dependency functions are exponentially decreasing ones. Polynomial and heuristic algorithms are presented for several versions of the problem.

In this paper the case of an epidemic spread in several distinct subpopulations in a region is examined. The problem of scheduling a single available resource (e.g. a mobile medical team or equipment), when there are several distinct areas where the subpopulations are infected, is considered here as a job-scheduling problem using the concept of deteriorating jobs, i.e. the model represents increasing loss rate as more susceptibles become infected, and increasing time and effort needed for the epidemic's containment. A deterministic model, appropriate for large populations, where random interactions can be averaged out, is used for the epidemic's rate of spread and an illustrative example of the developed B&B algorithm for solving the job-scheduling problem is presented

2. Statement of the problem

Allocating and scheduling resources for epidemic control is a very complex problem due to the fact that epidemics of infectious diseases is nonlinear and dynamic, since the rate of new infections is proportional to the product of the number of infected people and the number of uninfected people, and these quantities change over time. In a typical epidemic, the number of infected individuals vs. time follows an S-shaped curve.

The control actions in an epidemic model typically include vaccination of susceptibles (which increase the immunization rate), treatment or removal (e.g. quarantine) of infectious persons (which increases the removal rate) and reduction of the sufficient contact rate between susceptibles and infectious persons (average number of infecting contacts per infected person per unit time).

Notation

Let:

n : be the number of subpopulations infected simultaneously. Under an arbitrary epidemic's control sequence (permutation) of all $n!$ potential control sequences of the subpopulations' set $S=\{S_1, S_2, \dots, S_n\}$, denote the first subpopulation to be treated as $S_{[1]}$, the second as $S_{[2]}$, etc.

N_i : be the size of S_i .

$x_i(t)$: be the number of susceptible individuals in S_i at time t .

$y_i(t)$: be the number of infected individuals in S_i at time t .

$z_i(t)$: be the number of removed individuals in S_i at time t .

$\lambda_i(t)$: be the percentage rate of contacts sufficient to cause disease transmission at time t in S_i (sufficient contact rate).

$\mu_i(t)$: be the percentage rate at which susceptibles are immunized per unit time at time t in S_i .

$\kappa_i(t)$: be the percentage rate of removal (or “therapy”, e.g. quarantine) from the infected group in S_i at time t .

δ_i : be the percentage rate of entry into (and exit from) a group in S_i (replacement rate).

$t_0 > 0$: be the time required for the processor to commence controlling the epidemic in the first subpopulation.

$H_i(t)$: be the cumulative number of infections in S_i at time t .

$P_i(t)$: be the processing time, i.e. the time needed to control the epidemic, at time t in S_i .

C_i : be the completion time of controlling the epidemic in S_i .

Following the previous definitions, the problem is formulated as follows:

Find $\langle S_{\text{optimal}} \rangle = \langle S_{[1]}, S_{[2]}, \dots, S_{[n]} \rangle$ such that the total number of infections be minimized (or, equivalently, the total number of infections averted be maximized):

$$\min \sum_{i=1}^n H_i(C_i) \quad (1)$$

Assumptions:

- No interactions occur between control actions.
- Actions affecting a rate in one subpopulation do not affect the respective rate in another subpopulation.
- The replacement rate δ_i is the same for infected, susceptible and removed individuals and is assumed to be constant. This assumption is reasonable when death due to infection is inevitable but occurs “reasonably far” into the future. The assumption of a constant replacement rate is reasonable when the time horizon is relatively short ([3]).
- Each subpopulation’s size and control actions’ rates are constant, i.e. $\lambda_i(t) = \lambda_i$, $\mu_i(t) = \mu_i$, $\kappa_i(t) = \kappa_i$.
- All the infected individuals are assumed to be infectious.
- All infected individuals are equally likely to mix with all uninfected individuals (homogeneous mixing).
- The resources’ traveling times are assumed to be negligible.

3. The model

The model used is a simple deterministic compartmental Susceptible/Infective/Treated and/or Removed epidemic one (SIR) and it can be illustrated schematically in Figure 1 ([4]):

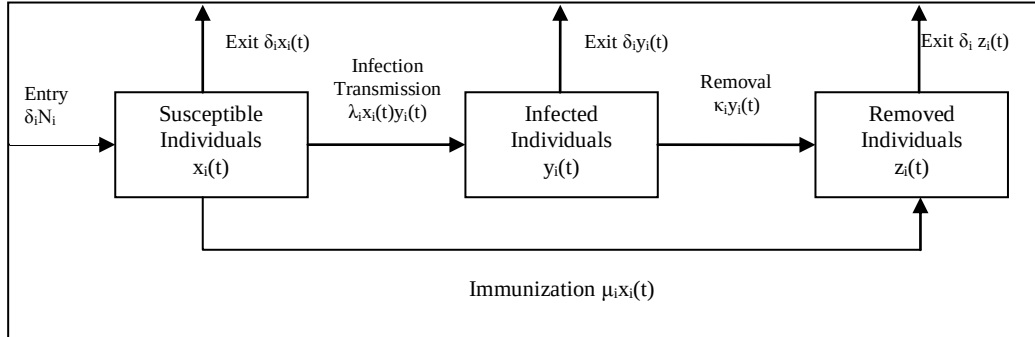


Figure 1. A simple compartmental epidemic model with control actions.

Each subpopulation size is considered to be constant. Individuals enter S_i at rate $\delta_i N_i$ and exit at rate $\delta_i(x_i(t)+y_i(t)+z_i(t))$. Susceptibles can become infected or immunized and infected individuals can be removed.

Thus the epidemic can be described by the following system of non-linear differential equations:

$$x_i'(t) = \delta_i N_i - \lambda_i x_i(t) y_i(t) - \delta_i x_i(t) - \mu_i x_i(t) \quad (2)$$

$$y_i'(t) = \lambda_i x_i(t) y_i(t) - \delta_i y_i(t) - \kappa_i y_i(t) \quad (3)$$

$$z_i'(t) = \mu_i x_i(t) + \kappa_i y_i(t) - \delta_i z_i(t) \quad (4)$$

$$x_i(t) + y_i(t) + z_i(t) = N_i \quad (5)$$

It is assumed, without loss of generality, that:

$y_i(0)=1$ and $z_i(0)=0$ (and consequently $x_i(0)=N_i-1$).

Equations (2)-(4) yield the rate of change in the numbers of susceptible, infective and removed individuals according to the entries and exits illustrated in Figure 1. Equation (5) ensures that S_i 's size is constant. In accordance to the law of mass action ([6], [10]), susceptibles are becoming infectives at a rate proportional to the product of the sizes of the two subpopulations. Usually the susceptibles' population is a non-increasing function and the infectives' population is a non-decreasing function. Therefore it is:

$$H_i(t) = \int_0^t \lambda_i x_i(\tau) y_i(\tau) d\tau \quad (6)$$

Solving the previous system of non-linear differential equations yields the following results:

$$x_i(t) = (N_i - 1 - B_i - \Gamma_i) e^{-\frac{\delta_i N_i}{\Gamma_i} t} + B_i e^{A_i t} + \Gamma_i \quad (7)$$

$$y_i(t) = e^{A_i t} \quad (8)$$

where A_i , B_i , Γ_i are constants depending on the previously mentioned constant rates λ_i , μ_i , κ_i , δ_i and N_i .

The control actions take place in parallel, thus the processing time function (control action time function) is:

$$P_i(t) = \max \left\{ \frac{1}{\mu_i x_i(t)}, \frac{1}{\kappa_i y_i(t)}, \frac{1}{\lambda_i x_i(t) y_i(t)} \right\}$$

and from Equations (6)-(8) the total number of new infections function is:

$$H_i(t) = \left(1 + \frac{\delta_i + \kappa_i}{A_i} \right) e^{A_i t} - 1$$

Consequently, after the elimination of the constant terms, the objective function (1) is:

$$\min \sum_{i=1}^n H_{[i]}(C_{[i]}) = \min \sum_{i=1}^n \frac{\delta_{[i]} + \kappa_{[i]}}{A_{[i]}} e^{A_{[i]} C_{[i]}}$$

where

$$C_{[i]} = \sum_{r=1}^i P_{[r]}$$

4. Solution methodology

The methodology employed to solve the problem consists of the following steps: A branch and bound algorithm is implemented so as to find a sequence of epidemic's control actions that yields the minimum total number of new infections in the affected subpopulations. Simple heuristic algorithms are implemented in order to calculate a satisfactory initial upper bound and a lower bound in each step of the elimination procedure. A dominance rule is also applied for curtailing the tree size.

B&B algorithm

Let the set of n sub-populations be divided into two subsets x and y . Let $\langle x \rangle$ be a sequence of sub-populations "serviced" in x . The values of $H_{[i]}$, $P_{[i]}$ and $C_{[i-1]}$ are known for every sub-population $S_{[i]} \in \langle x \rangle$ (the set of scheduled control actions). These have to be calculated for the subpopulations in set y (the set of unscheduled sub-populations "services"). The set $\langle x \rangle$ will be noted as a "node" in the B&B algorithm.

i. Extraction of upper bound: An initial upper bound can be calculated by scheduling the control actions in non decreasing order of A_i , $i=1,2,\dots,n$ (which is quite obvious from the form of the objective function where A_i are the exponent's coefficients).

ii. Extraction of lower bound: The following Lemma provides a lower bound for the B&B algorithm:

Lemma: Suppose that the epidemic's control of all scheduled subpopulations in a node is completed at time $C_{[r]}$, i.e. subpopulation $S_{[r]}$ is the last scheduled one. Then a lower bound of the optimal solution for this node is

$$LB = \sum_{i=1}^r H_{[i]}(C_{[i]}) + \sum_{j=1}^{|y|} (\delta_j + \kappa_j) A_j e^{-1} C_{[r]}^e$$

where $|y|$ is the number of the unscheduled subpopulations' epidemic control actions.

Proof: From the monotonicity of the function $\ln x/x$, it is quite easy to prove that $e^x \geq x^e$, $x \geq 0$.

Using this property, the lower bound can be easily derived from the objective function.

iii. Branching rule

Let X be the set of all possible sequences $\langle x \rangle$ (set of "active" nodes) in the B&B tree at any level. The node $\langle \text{active} \rangle \in X: |\text{active}| = \max |x|$, $x \in X$, is selected for branching. A greedy choice algorithm is applied for breaking ties, i.e. the node (at the same level of the tree) with the lowest lower bound is selected for branching.

iv. Exclusion principle- Dominance rule:

The current problem's solution consists of the best-known sequence at any time and the new upper bound, which is the total number of new infections when all the subpopulations' treatments are sequenced in a branch. Eliminate the sub-trees rooted at the nodes, for which their lower bound is greater than or equal to the upper bound of the objective function.

The following theorem provides some conditions under which all completed processes of a partial epidemic control schedule $\langle S_{s2} \rangle$ can be eliminated because a schedule at least as good exists among the completed processes of another partial epidemic control schedule $\langle S_{s1} \rangle$:

Theorem 1: If there exist two partial epidemic's control schedules $\langle S_{s1} \rangle$ and $\langle S_{s2} \rangle$, $S_{s1}, S_{s2} \subset S$ with: a) the same sets of subpopulations' indices with $|S_{s1}| = |S_{s2}| \geq 3$, b) the same last scheduled subpopulation $S_{[r]}$,

c) $\sum_{i \in S_{s1}} H_i(C_{[r]}) \leq \sum_{i \in S_{s2}} H_i(C_{[r]})$ and d) $C_{s1,[r]} \leq C_{s2,[r]}$ (the completion time of the last scheduled control in $\langle S_{s1} \rangle$ is less than its counterpart in $\langle S_{s2} \rangle$), then $\langle S_{s1} \rangle$ dominates $\langle S_{s2} \rangle$.

Proof: Similar to ([12]).

The implementation of the dominance rule is expected to have a quite significant impact on the overall algorithmic performance resulting in a smaller number of nodes to be evaluated and an average reduction in CPU time.

Finally, a special case where an exact solution can be yielded is illustrated in the following Theorem 2 (the proof can be obtained by a rather simple pair wise interchange argument):

Theorem 2: If there is a sequence of epidemics' controls with: a) non-decreasing A_i , b) non-decreasing $\frac{A_i + \delta_i + \kappa_i}{\lambda_i}$ and c) non-decreasing $\frac{\delta_i + \kappa_i}{A_i}$, $i=1,2,\dots,n$ then this sequence is optimal for the criterion of the total number of new infections.

5. Illustrative example

Consider four subpopulations for which data are given in Table 1. The parameters' values are similar to the ones' used in ([3]).

i	N_i	λ_i	μ_i	κ_i	δ_i	A_i	B_i	Γ_i
1	100	0,17	0,2	0,15	0,01	0,649524	0,94183	4,761905
2	65	0,15	0,1	0,08	0,01	0,796364	0,97793	5,909091
3	80	0,12	0,15	0,14	0,008	0,338076	0,97984	4,050633
4	70	0,14	0,08	0,02	0,009	0,962011	0,94291	7,078652

Table 1. Data for four subpopulations.

The B&B tree is depicted in Figure 2. Each node provides the partial epidemic's control schedule and its lower bound. The optimal sequence is $\langle S_1 S_4 S_2 S_3 \rangle$.

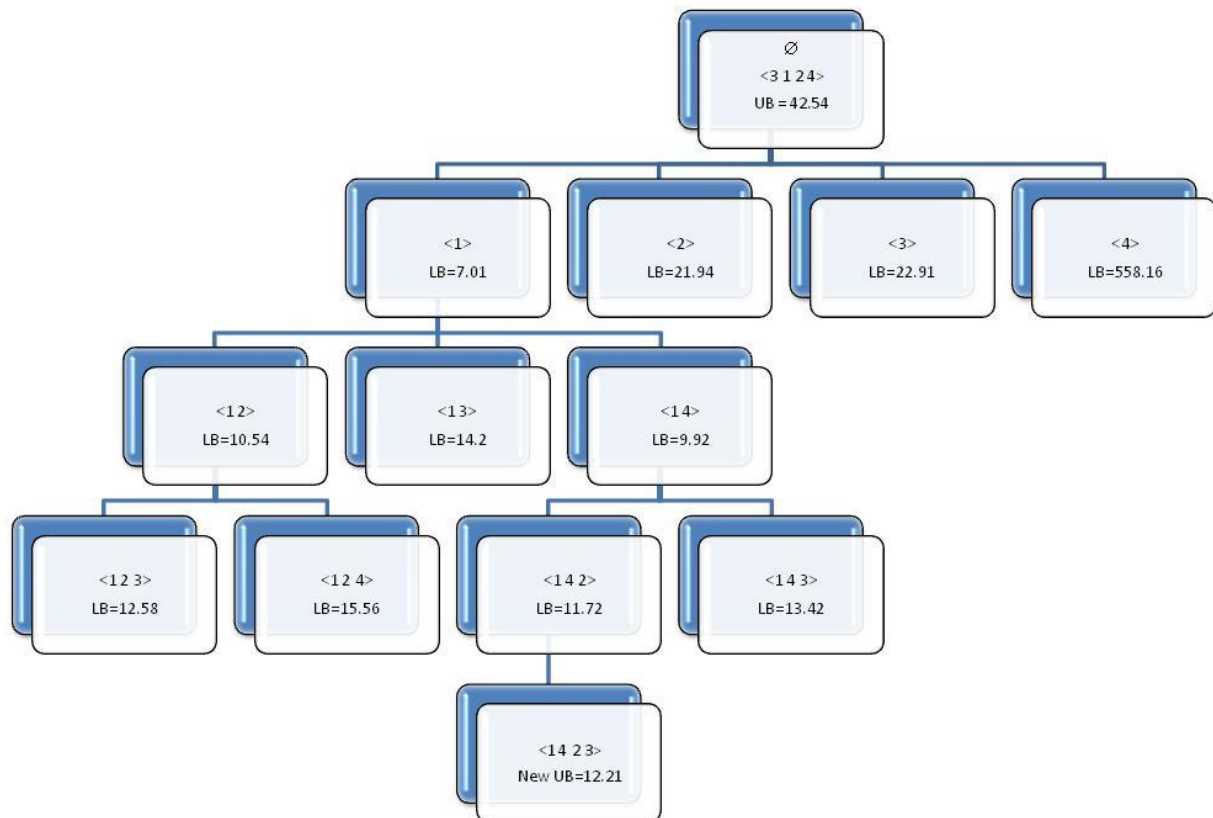


Figure 2. Illustrative B&B tree.

6. Conclusion

In an epidemic infection, resources to tackle it are usually limited, while time and effort required to control it are increasing functions of the starting time of the containment effort. The problem of scheduling a single available resource, when there are several areas where the population is infected, is considered. A deterministic model, appropriate for large populations, where random interactions can be averaged out, is used for the epidemic's rate of spread. The problem is tackled using the concept of deteriorating jobs, i.e. the model represents increasing rate of loss as more susceptibles become infected, and increasing time and effort needed for the epidemic's containment.

Comprehensive Emergency Management follows an all hazard approach, generalizing policies and plans for all kinds of emergencies, but different optimal approaches for different specific incidents may be found. The decision-making process (response policy) during a disaster response differs drastically from conventional decision-making. The problem's environment is changing rapidly and there is very little time for making a decision. Disaster operations management shows great potential as a new direction for OR/MS research. More specifically, epidemics have and will continue to appear and it is crucial to understand how they can be managed effectively and efficiently.

The existing analytical work on epidemic control focuses only on resource allocation and it has a limited applicability due to the simplicity of the epidemic models and simple control policies that are assumed. The next research steps of this paper include the heuristic and B&B algorithms' implementation in a real case study and their validation performing a sensitivity analysis, the examination of the time horizon's impact to the model and the incorporation of different actions' rates that depend on time. Future studies can also incorporate a more realistic epidemic model including features such as subdivision of the population by risk group and disease stage, non-homogeneous mixing, different types of infectious contacts, non-constant population sizes, different rates of exit from different groups and stochastic parameters. Finally the utilization of multiple resources in the model is an important topic for further investigation.

Acknowledgements

Dr. Nikolaos P. Rachaniotis's contribution to this research work has been supported by the Greek State Scholarships Foundation under grant number 851.

References

1. B. Alidaee, N.K. Womer. Scheduling with time dependent processing times: Review and extensions. *Journal of the Operational Research Society*, 50: 711-720 (1999).

2. N. Altay, W.G. Green III. OR/MS research in disaster operations management. *European Journal of Operational Research*, 175: 475-493 (2006).
3. M.L. Brandeau, G. S. Zaric, A. Richter. Resource allocation for control of infectious diseases in multiple dependent populations: beyond cost-effectiveness analysis. *Journal of Health Economics*, 22: 575-598 (2003).
4. M.L. Brandeau. Allocating Resources to Control Infectious Diseases. In Margaret L. Brandeau, Francois Sainfort and William P. Pierskalla (Eds.) "Operations Research and Health Care", Springer: 443-464 (2005).
5. TCE Cheng, Q. Ding, BMT Lin. A concise survey with time dependent processing times. *European Journal of Operational Research*, 152: 1-13 (2004).
6. D.J. Daley and J. Gani). *Epidemic Modelling: An Introduction*. Cambridge University Press (1999).
7. F. Fiedrich, F. Gehbauer, U. Rickers. Optimized resource allocation for emergency response after earthquake disasters. *Safety Science*, 35: 41-57 (2000).
8. A. Janiak, M.Y. Kovalyov. Scheduling in a contaminated area: a model and polynomial algorithms. *European Journal of Operational Research*, 173: 125-132 (2006).
9. A. Janiak, M.Y. Kovalyov. Scheduling jobs in a contaminated area: a model and heuristic algorithms. *Journal of the Operational Research Society*, 59: 977-987 (2008).
10. E.H. Kaplan, D.L. Craft, L.M. Wein. Emergency response to a smallpox attack: The case for mass vaccination. *PNAS*, 99 (16): 10935-10940 (2002).
11. C.P. Pappis, N.P. Rachaniotis. Scheduling in a Multi-Processor Environment with Deteriorating Job Processing Times and Decreasing Values: the Case of Forest Fires. *Proceedings of MISTA 2007, Paris, France, 28-31 August 2007*: 402-409 (2007).
12. C.P. Pappis, N.P. Rachaniotis. Scheduling a single fire fighting resource with deteriorating fire suppression times and set-up times. *Operational Research: An International Journal*, accepted for publication (2009).
13. N.P. Rachaniotis, C.P. Pappis. Scheduling Fire Fighting Tasks Using the Concept of "Deteriorating Jobs". *Canadian Journal of Forest Research*, 36: 652-658 (2006).
14. <http://www.ifrc.org>

Scheduling Inspection Operations subject to a Fixed Production Schedule

Boris Detienne · Stéphane Dauzère-Pérès ·
Claude Yugma

Abstract This paper deals with an optimization problem in which a set of inspection operations have to be processed so that a fixed production schedule is as reliable as possible. The problem is formalized as a parallel machine scheduling problem whose objective is to minimize a regular step total cost function. Mixed Integer Linear Programming models are described for the cases with and without release dates. Experimental results are reported and discussed to evaluate the performance of a standard mathematical programming solver applied to different special cases and to the general case.

1 Introduction

Let us consider a production process in which lots of various products types are scheduled on several production machines. When they are completed, some of these lots are selected to be inspected on several specialized machines, to check whether the process is out of the specifications for the corresponding product type. If this is the case, the production of the lots of the same product type should be stopped or adjusted. Hence, it is important to react as soon as possible to avoid producing lots that may be scrapped, i.e. to inspect lots before too many lots are produced.

Hence, we are interested in solving an optimization problem in which the scheduling of inspection operations has to be determined, so that a set of fixed (already scheduled) production operations are processed with the highest possible quality. The inspection operation of a given lot is associated with the lots of the same type that must be produced. As already mentioned, there is a risk of scrapping a lot if its production starts before the lot of the same product type is inspected. The objective of the problem is thus to minimize the overall risk incurred by production lots, by optimally scheduling the set of inspection operations.

This problem arises from an actual industrial example. Semiconductor manufacturing involves many sensitive operations of lots of (usually 25) wafers, in which a slight

École Nationale Supérieure des Mines de Saint-Étienne
Centre Microélectronique de Provence, Site Georges Charpak
880 route de Mimet, F-13541 Gardanne, France
E-mail: detienne@emse.fr, dauzere-peres@emse.fr, yugma@emse.fr

change in the processing conditions may result in the loss of some or all wafers of the lot. A strategy commonly used in this situation consists in frequently adjusting the production equipment parameters on the basis of measurements made on products. The limited capacity of inspection resources (here, metrology tools) lead to restrict the number of measures, so that one measurement operation is performed for the production of a few, typically less than ten, consecutive lots. After this number of lots has been processed, further production of lots of the same type on the same machine is considered more or less risky until a measurement operation is performed and the equipment parameters adjusted according to the results of the measure. However, the extremely high costs of the production machines lead the decision-maker to continue producing without allowing any idle time. In this context, it becomes particularly important to schedule efficiently the measurement operations in order to match the production schedule as well as possible.

2 Problem statement

2.1 Machine scheduling formulation

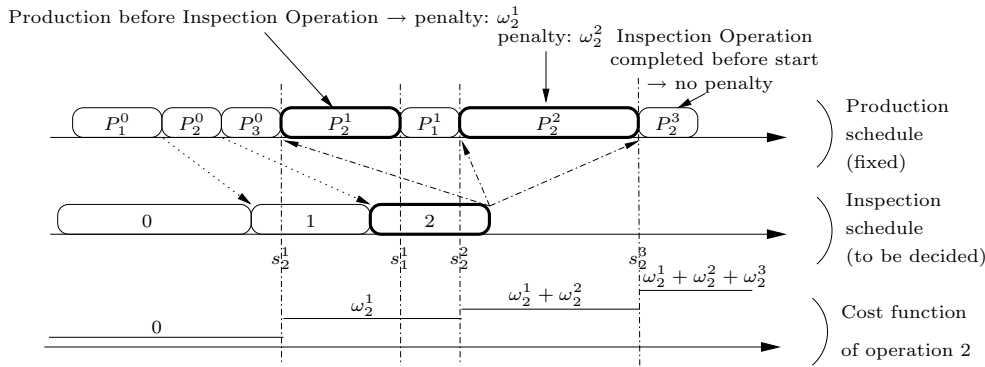


Fig. 1 Illustration of the problem with one production machine and one inspection machine

Figure 1 illustrates a simple case of our problem, where inspection operations 1 and 2 are made on the lots processed in production operations P_1^0 and P_2^0 . The results of the inspection operations are meant to be used for adjusting process parameters for, respectively, production operations $\{P_1^1\}$ and $\{P_2^1, P_2^2, P_2^3\}$. On the given inspection equipment schedule, product type 1 completes before the beginning of P_1^1 , which is then safely executed. However, the inspection of product type 2 ends after production operations P_2^1 and P_2^2 start. These operations then incur a risk of producing products that will not meet quality specifications. As the production schedule is fixed, the total risk incurred by a set of lots related to a given inspection operation is determined by its completion time: if 2 ends before the start of P_2^1 (at s_2^1), then the risk is zero ; if it completes between s_2^1 and s_2^2 , then only P_2^1 is not safe and the risk associated to 2 is equal to the expected cost ω_2^1 for executing P_2^1 before the inspection operation is completed. When completing between s_2^2 and s_2^3 , 2 is ready only for P_2^3 , and the expected cost is $\omega_2^1 + \omega_2^2$...

Thus, the evaluation of the total risk not prevented by an inspection operation can be represented by a non-decreasing step function of its completion time. The overall problem consists in minimizing the total cost of the schedule. As a consequence, any regular total cost function of integer completion times can be induced by a proper production schedule, so that the problem is clearly strongly NP-hard. This paper focuses on the cases where inspection operations are performed frequently, i.e. cost functions only have a few steps. To the best of our knowledge, the existing methods which can be applied to this problem are developed for the general total cost machine scheduling problem (see for example [5] or [7]). These approaches, tested mainly on the weighted tardiness problem, exploit dominance rules based on job interchanges which may not be so active in our case because of the high degeneracy induced by the stepwise structure of the cost function.

The main contribution of the paper is to propose models bypassing this degeneracy and that could be used for future work on this problem. In order to provide an idea *a priori* of the relevance of methods based on these formulations, we report the performance of a MILP solver with a standard strategy.

2.2 Mathematical formulation

The scope of this paper comprises the case in which inspection operations have to be scheduled on parallel unrelated machines, with release dates constraints, as well as some special cases. Let us now define the most general problem investigated formally.

A set J of jobs (inspection operations) have to be scheduled on a set M of machines (inspection machines). Job $j \in J$ cannot start before its release date r_j . Let us denote p_j^μ the duration of job j processed on machine μ , and $\Theta_j = (s_j^1, \dots, s_j^{q_j})$ the set of starting times of the $q_j - 1$ production operations associated with j ; by convention, the last jump point $s_j^{q_j}$ of j corresponds to the end of the scheduling horizon $T = \{1, \dots, \tau\}$ (we can set $\tau = \max_j r_j + \sum_j \max_\mu p_j^\mu$). We assume, without loss of generality, that $s_j^1 < s_j^2 < \dots < s_j^{q_j-1} < s_j^{q_j}$. In the sequel, we will mention these dates as *jump points* associated to j . The cost of completing job j in $\{s_j^{i-1} + 1, \dots, s_j^i\}$ is denoted by ω_j^i . The overall cost of a solution is equal to the sum of the costs associated to each job.

Time-indexed model Let us denote c_{jt} the cost of completing job j at time instant t . According to the previous notations, we have $c_{jt} = \omega_j^k$, with k such that $s_j^{k-1} < t \leq s_j^k$. Let us introduce the set of decision variables $(x_{jt}^\mu)_{j \in J, t \in T, \mu \in M}$, such that x_{jt}^μ is equal to 1 iff j completes at t on processor μ , 0 otherwise. This allows us to write the classical time-indexed formulation of the problem:

$$(TI) : \min \sum_{\mu \in M} \sum_{j \in J} \sum_{t \in T} c_{jt} x_{jt}^\mu \quad (1)$$

$$s.t. \sum_{\mu \in M} \sum_{t=r_j+p_j^\mu}^T x_{jt}^\mu = 1 \quad j \in J \quad (2)$$

$$\sum_{j \in J} \sum_{\theta=t+1}^{t+p_j^\mu} x_{j\theta}^\mu \leq 1 \quad \mu \in M, t \in T \quad (3)$$

$$x_{jt}^\mu \in \{0, 1\} \quad \mu \in M, j \in J, t \in T \quad (4)$$

In this model, constraints (2) ensure that each job is processed exactly once, and constraints (3) ensure that no more than one job is executed at a time on each machine.

3 Special case : no release dates

This section presents a MILP formulation for the special case with no release dates. In practice, this problem can represent the scheduling of a set of inspection operations on lots already processed and present in a queue.

The MILP formulation presented below relies on the following well-known result (see e.g. [4]). Let us consider the problem of scheduling a set of jobs I on a single processor and without preemption, such that each job $i \in I$ completes before its deadline $\bar{d}_i$, with no release date restrictions.

Proposition 1 *If there exists at least one feasible schedule in which each job completes before its deadline, then the schedules where jobs are sequenced according to a non-decreasing order of their deadlines are feasible.*

This leads to the following characterization of feasible solutions.

Proposition 2 *Let σ be the permutation of I corresponding to a non-decreasing order of the deadlines. There exists at least one feasible solution iff:*

$$\sum_{r=1}^i p_{\sigma(r)} \leq \bar{d}_{\sigma(i)} \quad i \in I$$

3.1 MILP Formulation

The problem of scheduling inspection operations can clearly be seen as the problem of finding a minimum cost assignment of deadlines (corresponding to the jump points, where the cost functions change), such that there exists at least one feasible schedule meeting the deadlines. Using Proposition 2 for characterizing the set of feasible schedules leads to the following MILP model.

For the sake of readability, we introduce additional notations. Each jump point s_j^i , $j \in J$, $i \in \Theta_j$, is linked with a possible occurrence k of job j subject to a deadline d_k and whose execution cost is $\gamma_k = \omega_j^i$, corresponding to the execution of j before $d_k = s_j^i$. The set of occurrences associated with j is denoted K_j . Moreover, we note $K = \cup_{j \in J} K_j$. Let σ be a function from $\{1, \dots, |K|\}$ into J , giving the job corresponding to the occurrence located at each position of a non-decreasing order of the deadline of the occurrences.

$$(QTS) : \min \sum_{\mu \in M} \sum_{k \in K} \gamma_k \cdot u_k^\mu \quad (5)$$

$$s.t. \sum_{\mu \in M} \sum_{k \in K_j} u_k^\mu = 1 \quad j \in J \quad (6)$$

$$\sum_{l=1}^k p_{\sigma(l)}^\mu \cdot u_l^\mu \leq d_k \quad \mu \in M, k \in K \quad (7)$$

$$u_k^\mu \in \{0, 1\} \quad \mu \in M, k \in K \quad (8)$$

In this model, decision variable u_k^μ is equal to 1 if and only if occurrence k is processed on machine μ . Constraints (6) ensure that exactly one occurrence of each job is processed, and constraints (7) ensure that this selection of occurrences can lead to a feasible schedule. This model can be seen as a generalization of a model used in the literature for the minimization of the weighted number of tardy jobs, e.g. in [6].

3.2 Experimental results

All tests were performed on a Personal Computer equipped with 3 GB RAM and a 2.5 GHz Intel Xeon processor, using the MILP solver XPRESS-MP 2007 with standard settings.

In order to build our test bed, we used a random instance generator that takes as input a number of jobs n , a number of processors m , a number of jump points per job K , and a tardiness factor T . For each of the n jobs, the generator draws m durations from a uniform distribution in $\{1, \dots, 100\}$, K jump points in $\{p_j^1, \dots, \frac{nT}{m}\}$, and a cost increase in $\{1, \dots, 100\}$ for each jump point. For each value of n and m , 5 instances were generated for each combination of parameters $T \in \{10, 20, 50\}$ and $K \in \{2, 3, 4, 9\}$. In the industrial application presented in Section 1, these values of K correspond respectively to the measurement of one lot every 3, 4, 5 and 10 lots.

	n									
K	10	20	30	50	100	200	300	400	500	Total
2	15	15	15	15	15	15	15	15	15	135/135
3	15	15	15	15	15	15	14	14	3	121/135
4	15	15	15	15	15	15	8	0	0	98/135
9	15	15	15	15	9	0	0	0	0	69/135
Total	60/60	60/60	60/60	60/60	54/60	45/60	37/60	29/60	18/60	423/540

Table 1 Number of single machine instances solved optimally within 1000 sec.

	n								
K	10	20	30	50	100	200	300	400	500
2	0.0	0.0	0.1	0.2	0.5	3.4	10.9	22.7	48.8
3	0.0	0.0	0.1	0.5	4.2	44.7	161.0*	395.5*	641.9*
4	0.0	0.2	0.3	1.3	16.2*	201.9	350.5*	+	+
9	0.1	0.7	3.8	23.7	388.7*	+	+	+	+

Table 2 Average computing time for single machine instances solved optimally (sec.). *Not all instances were solved within the 1000 sec. time limit. +No instance was solved within the time limit.

We first report the results obtained using the standard solver XPRESS-MP 2007 in the case of a single processor. Tables 1 and 2 give the number of instances solved by the solver within a time limit of 1000 seconds, and the average computing time for the instances that are solved optimally. One can observe that, as expected, the number of jump points directly impacts the results, since it is linked to the size of the MILP.

Under the same conditions, the solver applied on the time-indexed model solves less than 25% of the 30-job instances. It can easily be explained by the number of variables, as well as the degeneracy due to the number of variables with the same cost, present in this model. Tables 3 and 4 report the performance of the solver on parallel unrelated machines, within the same time limit of 1000 seconds. These results show that problems with up to 50 jobs and a *sampling rate* of $\frac{1}{5}$ can be solved optimally without further investigation, in an operational computing time.

	n					
K \ m	10	20	30	50	100	Total
K=2	60/60	60/60	60/60	60/60	58/60	298/300
m=2	15	15	15	15	15	75/75
m=3	15	15	15	15	15	75/75
m=6	15	15	15	15	13	73/75
m=9	15	15	15	15	15	75/75
K=3	60/60	60/60	60/60	60/60	48/60	288/300
m=2	15	15	15	15	14	74/75
m=3	15	15	15	15	12	72/75
m=6	15	15	15	15	10	70/75
m=9	15	15	15	15	12	72/75
K=4	60/60	60/60	60/60	60/60	38/60	278/300
m=2	15	15	15	15	14	74/75
m=3	15	15	15	15	8	68/75
m=6	15	15	15	15	6	66/75
m=9	15	15	15	15	10	70/75
K=9	60/60	60/60	60/60	44/60	8/60	232/300
m=2	15	15	15	10	1	56/75
m=3	15	15	15	8	0	53/75
m=6	15	15	15	11	2	58/75
m=9	15	15	15	15	5	65/75
Total	240/240	240/240	240/240	224/240	152/240	1096/1200

Table 3 Number of parallel unrelated machine instances solved optimally within 1000 sec.

Well-known special case: $R||\sum w_j U_j$ Adapting the model (*QTS*) to this problem is straightforward. Hence, it corresponds to the special case where $q_j = 2$, for all $j \in J$ (the parameter K of the generator is set to 1). This special case of the model already proved to be of practical use, since M'Hallah and Bulfin [6] published a method that relies on a similar formulation and is capable of solving exactly instances with up to 400 jobs and 8 machines. To the best of our knowledge, it is currently the best method published for $R||\sum w_j U_j$.

The direct application of XPRESS-MP on (*QTS*) is not so efficient, although all tested instances with 500 jobs and 2 machines are solved under a time limit of 1000 seconds, with an average time of 22 seconds. However, it does not perform so well for the case with 8 machines, since only less than one half of the 400-job instances are solved within the time limit.

The results obtained by M'Hallah and Bulfin [6] and our experiments on (*QTS*) suggest that this model is a good basis for designing dedicated methods for our problem.

4 General case

This section presents a model for the general case with release dates of the problem of scheduling inspection operations. The formulation relies on the following strict order among the set of job occurrences.

Definition 1 Let $i \in J$, $j \in J$, $k_i \in K_i$, and $k_j \in K_j$. We define the (strict partial) order $\prec$ by $d_{k_i} \prec d_{k_j}$ if one of these two relations hold:

$$d_{k_i} < d_{k_j} \quad (9)$$

$$(d_{k_i} = d_{k_j}) \wedge (r_i < r_j) \quad (10)$$

	n				
K / m	10	20	30	50	100
K=2	0.0	0.1	0.3	8.9	84.2
m=2	0.0	0.1	0.3	1.1	12.1
m=3	0.0	0.1	0.3	4.1	37.1
m=6	0.0	0.0	0.3	30.1	167.4*
m=9	0.0	0.0	0.1	0.5	131.3
K=3	0.0	0.2	0.8	31.5	209.8
m=2	0.0	0.3	0.9	3.1	168.0*
m=3	0.0	0.3	1.7	43.1	329.0*
m=6	0.0	0.0	0.6	78.5	244.4*
m=9	0.0	0.0	0.1	1.2	110.7*
K=4	0.0	0.4	3.0	62.2	228.9
m=2	0.0	0.4	1.7	18.8	354.8*
m=3	0.0	1.0	9.2	108.0	285.9*
m=6	0.0	0.2	0.9	120.3	161.9*
m=9	0.0	0.1	0.2	1.8	47.4*
K=9	0.1	3.0	74.5	226.4*	479.1*
m=2	0.2	4.6	32.3	364.4*	913.1*
m=3	0.2	6.8	163.9	410.3*	+
m=6	0.0	0.5	101.1	264.6*	642.0*
m=9	0.0	0.2	0.8	8.3	327.1*

Table 4 Average computing time for parallel unrelated machine instances solved optimally (sec.).*Not all instances were solved within the 1000 sec. time limit.+No instance was solved within the time limit.

We propose to redefine the set of jump points of each job i by adding L_i , a set of virtual jump points given by:

$$L_i = \{d_{k_j} | \exists k_i \in K_i, \exists j \in J, \exists k_j \in K_j, (r_i < r_j) \wedge (d_{k_j} < d_{k_i})\} \quad i \in J$$

$$\gamma_{l_i} = \arg \min\{\gamma_k | k \in K_i, d_k \geq d_{l_i}\} \quad i \in J, l_i \in L_j$$

Hence, by construction, if a job i completes at C_i , before job j completing at $C_j \leq d_{k_j}$, then there always exists one jump point $e_i^{k_j} \in K_i \cup L_i$ such that $e_i^{k_j} \geq C_i$ and $e_i^{k_j} < d_{k_j}$.

The following propositions lead to a property of optimal schedules of a subset I of n_I jobs on a single machine. For the sake of readability, $p_j, j \in I$, denotes the processing time of job j on the machine considered.

First, let us recall that the set of active schedules (where jobs are scheduled as soon as possible) is dominant for our problem, since all job cost functions are regular. So, any optimal schedule of a subset of jobs can be represented by an ordered sequence of these jobs.

Let us introduce the following notations, associated with any sequence of jobs δ :

- Let $\delta(s)$ denote the job at position s in δ ,
- Let $S(\delta)$ denote the schedule obtained by scheduling the sequence of jobs δ as soon as possible, in the order they appear in δ ,
- Let $C_j(\delta), j \in I$, denote the completion time of j in $S(\delta)$,
- Let $f_j(\delta), j \in I$, denote the cost of j in $S(\delta)$,
- Define $K(\delta) = (k_1(\delta), \dots, k_{n_I}(\delta))$ such that $k_j(\delta) = \arg \min\{d_k | k \in K_j \cup L_j, d_k \geq C_j(\delta)\}, j \in I$. Clearly, $k_j(\delta)$ is the jump point immediately succeeding job j in $S(\delta)$.

Proposition 3 Let us consider a sequence of jobs δ , $i = \delta(s)$ and $j = \delta(s+1)$, with $s < |\delta|$. If $d_{k_i(\delta)} \prec d_{k_j(\delta)}$ does not hold, then $(d_{k_i(\delta)} \geq d_{k_j(\delta)})$ and $(r_i \geq r_j)$.

Proof The logical negation of $d_{k_i(\delta)} \prec d_{k_j(\delta)}$ is:

$$\begin{aligned} & (d_{k_i} \geq d_{k_j}) \wedge ((d_{k_i} \neq d_{k_j}) \vee (r_i \geq r_j)) \\ \Leftrightarrow & (d_{k_i} > d_{k_j}) \vee ((d_{k_i} \geq d_{k_j}) \wedge (r_i \geq r_j)) \\ \Leftrightarrow & ((d_{k_i} > d_{k_j}) \wedge (r_i \geq r_j)) \vee ((d_{k_i} > d_{k_j}) \wedge (r_i < r_j)) \vee ((d_{k_i} \geq d_{k_j}) \wedge (r_i \geq r_j)) \\ \Leftrightarrow & ((d_{k_i} > d_{k_j}) \wedge (r_i < r_j)) \vee ((d_{k_i} \geq d_{k_j}) \wedge (r_i \geq r_j)) \end{aligned}$$

Let us assume that $r_i < r_j$. Since i completes before j , there exists at least one jump point $e_i^{k_j(\delta)} \in K_i \cup L_i$ such that $C_i(\delta) \leq e_i^{k_j(\delta)} \leq d_{k_j(\delta)}$.

Thus, $((d_{k_i(\delta)} > d_{k_j(\delta)}) \wedge (r_i < r_j))$ is inconsistent. Hence, the claimed result holds. $\square$

Proposition 4 Let us consider a sequence of jobs δ , $i = \delta(s)$ and $j = \delta(s+1)$, with $s < |\delta|$. Define δ' as the sequence obtained from δ by interchanging i and j , i.e. $\delta'(s) = j$, $\delta'(s+1) = i$ and $\delta'(x) = \delta(x)$, $x \in \{1, \dots, |\delta|\} - \{s, s+1\}$. Moreover, let $t_0 = 0$ if $s = 1$, or $t_0 = C_{\delta(s-1)}(\delta)$ otherwise.

If $(r_i \geq r_j)$, then $C_i(\delta') \leq C_j(\delta)$.

Proof Let us consider the following cases:

- $t_0 \geq r_i$: Then $t_0 \geq r_j$ and $C_i(\delta') = t_0 + p_j + p_i$. Moreover, $C_j(\delta) = t_0 + p_i + p_j$. Thus, $C_i(\delta') = C_j(\delta)$.
- $t_0 < r_i$: Then $C_i(\delta') = \max(r_i + p_i, C_j(\delta') + p_i)$. Since $S(\delta)$ is feasible, we have $r_i + p_i + p_j \leq C_j(\delta)$, and thus $r_i + p_i < C_j(\delta)$. Moreover, $C_j(\delta') = \max(t_0, r_j) + p_j$. Since $t_0 < r_i$ and $r_j \leq r_i$, $C_j(\delta') + p_i \leq r_i + p_i + p_j \leq C_j(\delta)$. Thus, $C_i(\delta') \leq C_j(\delta)$. $\square$

Proposition 5 Let us consider a sequence of jobs δ , $i = \delta(s)$ and $j = \delta(s+1)$, with $s < |\delta|$. Define δ' as the sequence obtained from δ by interchanging i and j , i.e. $\delta'(s) = j$, $\delta'(s+1) = i$ and $\delta'(x) = \delta(x)$, $x \in \{1, \dots, |\delta|\} - \{s, s+1\}$.

If $(d_{k_i(\delta)} \geq d_{k_j(\delta)})$ and $(r_i \geq r_j)$, then $f_i(\delta') \leq f_i(\delta)$.

Proof According to Proposition 4, $C_i(\delta') \leq C_j(\delta)$. Since $S(\delta)$ is feasible, $C_j(\delta) \leq d_{k_j(\delta)}$ and $C_i(\delta') \leq d_{k_j(\delta)} \leq d_{k_i(\delta)}$. Since the job cost functions are regular, the claimed result follows. $\square$

Proposition 6 Let us consider a sequence of jobs δ , $i = \delta(s)$ and $j = \delta(s+1)$, with $s < |\delta|$. Define δ' as the sequence obtained from δ by interchanging i and j , i.e. $\delta'(s) = j$, $\delta'(s+1) = i$ and $\delta'(x) = \delta(x)$, $x \in \{1, \dots, |\delta|\} - \{s, s+1\}$.

If $(r_i \geq r_j)$, then $f_j(\delta') \leq f_j(\delta)$.

Proof Since $r_i \geq r_j$, $C_j(\delta) = C_i(\delta) + p_j = \max(t_0, r_i) + p_i + p_j$. So $C_j(\delta') = \max(t_0, r_j) + p_j < C_j(\delta)$. Since the job cost functions are regular, the claimed result follows. $\square$

Proposition 7 Let us consider a sequence of jobs δ , $i = \delta(s)$ and $j = \delta(s+1)$, with $s < |\delta|$. Define δ' as the sequence obtained from δ by interchanging i and j , i.e. $\delta'(s) = j$, $\delta'(s+1) = i$ and $\delta'(x) = \delta(x)$, $x \in \{1, \dots, |\delta|\} - \{s, s+1\}$.

If $(r_i \geq r_j)$, then $f_{\delta(x)}(\delta') \leq f_{\delta(x)}(\delta)$, $x \geq s+2$.

Proof Clearly, the proposition only applies if $s + 2 \leq |\delta|$. Besides, by definition of $S(\cdot)$, $C_{\delta(s+2)}(\delta) = \max(C_j(\delta), r_{\delta(s+2)} + p_{\delta(s+2)})$, and $C_{\delta'(s+2)}(\delta') = \max(C_i(\delta'), r_{\delta'(s+2)} + p_{\delta'(s+2)})$. As $\delta(s+2) = \delta'(s+2)$ and $C_i(\delta') \leq C_j(\delta)$ (from Proposition 4), $C_{\delta(s+2)}(\delta') \leq C_{\delta(s+2)}(\delta)$. By recurrence, it is clear that $C_{\delta(x)}(\delta') \leq C_{\delta(x)}(\delta)$, $x \geq s + 2$.

Since the job cost functions are regular, the claimed result follows. $\square$

Proposition 8 *Let us consider an optimal sequence of jobs δ , $i = \delta(s)$ and $j = \delta(s+1)$, with $s < |\delta|$. Define δ' as the sequence obtained from δ by interchanging i and j , i.e. $\delta'(s) = j$, $\delta'(s+1) = i$ and $\delta'(x) = \delta(x)$, $x \in \{1, \dots, |\delta|\} - \{s, s+1\}$.*

Then, at least one of these two relation holds:

- (i) $d_{k_i(\delta)} \prec d_{k_j(\delta)}$
- (ii) *The cost of $S(\delta')$ is not greater than the cost of $S(\delta)$.*

Proof Let us assume that $\neg(i)$ holds, and let us prove that (ii) necessarily holds in this case.

First, according to Proposition 3, $(d_{k_i(\delta)} \geq d_{k_j(\delta)})$ and $(r_i \geq r_j)$. Then, Propositions 5, 6 and 7 imply $f_x(\delta') \leq f_x(\delta)$, $x \geq s$. Moreover, since δ and δ' are identical before the s^{th} position, we clearly have $f_x(\delta') = f_x(\delta)$, $1 \leq x < s$.

It follows that $\sum_{1 \leq x \leq |\delta|} f_x(\delta) \leq \sum_{1 \leq x \leq |\delta|} f_x(\delta')$. $\square$

Proposition 8 implies that the set of sequences of jobs following the $\prec$ -order is dominant for our problem.

4.1 MILP formulation

Proposition 8 shows that there exists at least one optimal solution that can be described by a selection of one jump point and one machine for each job. The corresponding schedule can be obtained by sequencing the jobs according to the order $\prec$ of the jump points, on each machine. Let us consider a sequence of the occurrences such that the associated deadlines satisfy the order $\prec$, and let σ be a function from $\{1, \dots, |K| + |L|\}$ into J , giving the job corresponding to the occurrence located at each position of a non-decreasing order of the deadlines of the occurrences. This allows us to write the following model where decision variables $(u_k^\mu)_{k \in K \cup L, \mu \in M}$ aim at selecting one jump point and one machine per job, and $(t_k^\mu)_{k \in K \cup L, \mu \in M}$ represent the job completion times associated to selected jump points and machines. These variables allow the expression of the conjunctive constraints as well as the respect of the deadlines of the job occurrences.

$$(QTS^\prec) : \min \sum_{\mu \in M} \sum_{k \in K \cup L} \gamma_k \cdot u_k^\mu \quad (11)$$

$$\sum_{\mu \in M} \sum_{k \in K_j \cup L_j} u_k^\mu = 1 \quad j \in J \quad (12)$$

$$t_k^\mu \geq t_{k-1}^\mu + p_{\sigma(k)}^\mu \cdot u_k^\mu \quad k \in K \cup L \quad (13)$$

$$t_k^\mu \geq (r_{\sigma(k)} + p_{\sigma(k)}^\mu) \cdot u_k^\mu \quad k \in K \cup L \quad (14)$$

$$t_k^\mu \leq d_k \quad k \in K \cup L \quad (15)$$

$$u_k^\mu \in \{0, 1\} \quad \mu \in M, k \in K \cup L \quad (16)$$

$$t_k^\mu \geq 0 \quad \mu \in M, k \in K \cup L \quad (17)$$

In this formulation, constraints (12) ensure that exactly one occurrence is selected for each job. (13) deal with the respect of the conjunctive constraints, (14) with the respect of the release dates, and (15) with the respect of the deadlines. This model can be seen as a generalization of the *master sequence* model, used by Dautère-Pères and Sevaux [3] for minimizing the weighted number of tardy jobs on a single machine.

Dominances The following proposition is the generalization of a dominance rule described in [3]. We do not provide a proof since they are closely related.

Proposition 9 *Let $i \in J$ and $j \in J - \{i\}$, $k_i \in K_i \cup L_i$, $k_j \in K_j \cup L_j$, and note by $k_j^+ \in K_j \cup L_j$ the jump point of j following immediately k_j . Assume that all conditions $r_i < r_j$, $r_i + p_i^\mu \geq r_j + p_j^\mu$, $r_i + p_i^\mu + p_j^\mu > d_{k_j}$, $r_j + p_j^\mu + p_i^\mu > d_{k_i}$, $d_{k_i} - p_i^\mu \leq d_{k_j} - p_j^\mu$ and $\gamma_{k_j^+} \geq \gamma_{k_j} + \max_{l \in K_i} \gamma_l$ hold. If i is processed on machine μ and completes before d_{k_i} , and j completes after d_{k_j} , then i can be replaced by j on μ and processed somewhere else such that the cost of the schedule does not increase.*

In a single machine environment, the proposition allows the suppression of all jump points of K_i prior to d_{k_i} . In the parallel machine case, the proposition can be interpreted as follows: in any optimal schedule, either i is processed on μ and completes before d_{k_i} and j is processed on another machine at a cost smaller than γ_{k_j} , or i is processed on another machine or completes after d_{k_i} . If all conditions stated in Proposition 9 hold, then the following inequality is valid and can be added to the model:

$$\sum_{l_i \in K_i \cup L_i | d_{l_i} \leq d_{k_i}} u_{l_i}^\mu \leq \sum_{\mu' \in M - \{\mu\}} \sum_{l_j \in K_j \cup L_j | \gamma_{l_j} \leq \gamma_{k_j}} u_{l_j}^{\mu'}$$

The size of the model is directly linked to the overall number of jump points. Thus, it seems profitable to avoid adding as many virtual points as possible. Immediate rules prevent from adding useless points: if $r_i < r_j$, $d_{k_j} < d_{k_i}$ and $r_i + p_i^\mu + p_j^\mu > d_{k_j}$, then the virtual jump point corresponding to the execution of i before j such that j completes before d_{k_j} on μ is useless since this configuration is infeasible. Moreover, if k_i^- denotes the jump point of i immediately preceding k_i and $d_{k_i^-} + p_i^\mu > d_{k_j}$ then, in any feasible schedule where i is processed before j and j completes before d_{k_j} , there already exists a jump point between the completion time of i and d_{k_j} , and the corresponding virtual jump point is useless. Besides, it is in some cases always possible to process i between $d_{k_j} + 1$ and d_{k_i} without increasing the cost of the schedule. Considering the case where i is before j , and j completes before d_{k_j} is then not mandatory. Such configurations can be detected by solving an auxiliary $1|r_j|L_{max}$ problem with appropriate release and due dates [2].

4.2 Experimental results

In order to build our test bed, we used an additional parameter R to the instance generator presented in Section 3.2. One release date per job is drawn from a uniform distribution $\{0, \dots, \frac{nR}{m}\}$, and jump points are drawn from $\{r_j + p_j^1, \dots, \frac{nT}{m}\}$. The values tested for R are $R \in \{5, 10, 20\}$. The benchmarking conditions are the same as for the case without release dates, except that a time limit of 1000 seconds is used. Under these conditions, the solver applied to the time-indexed formulation solved only a few 30-job instances.

Tables 5 and 6 report the results obtained by applying XPRESS-MP 2007 with a standard strategy on the model ($QTS^{\prec}$) on a single machine, while tables 7 and 8 give the results for the unrelated parallel machine case. We can see that this straightforward approach for solving the problem is suitable for instances with up to 30 jobs or high sampling rates ($K = 2$, $K = 3$). However, it is clearly limited to more complex instances.

k \ n	10	20	30	50	Total
2	45	45	45	43	178/180
3	45	45	45	42	177/180
4	45	45	45	31	166/180
9	45	45	32	8	130/180
Total	180/180	180/180	167/180	124/180	651/720

Table 5 Number of single machine instances with release dates solved optimally within 1000 sec.

k \ n	10	20	30	50
2	0.0	0.2	0.8	8.3*
3	0.1	0.5	4.2	38.5*
4	0.1	1.1	13.7	190.6*
9	0.5	60.8	142.6*	503.3*

Table 6 Average computing time for single machine instances with release dates solved optimally (sec.). *Not all instances were solved within the 1000 sec. time limit. +No instance was solved within the time limit.

	n				
k \ m	10	20	30	40	Total
k=2	135/135	135/135	133/135	121/135	524/540
m=2	45	45	44	42	176/180
m=3	45	45	44	35	169/180
m=6	45	45	45	44	179/180
k=3	135/135	135/135	133/135	102/135	505/540
m=2	45	45	44	34	168/180
m=3	45	45	44	27	161/180
m=6	45	45	45	41	176/180
k=4	135/135	135/135	119/135	82/135	471/540
m=2	45	45	36	18	144/180
m=3	45	45	38	23	151/180
m=6	45	45	45	41	176/180
k=9	135/135	132/135	80/135	43/135	390/540
m=2	45	44	17	4	110/180
m=3	45	43	20	5	113/180
m=6	45	45	43	34	167/180
Total	540/540	537/540	465/540	348/540	1890/2160

Table 7 Number of parallel unrelated machine instances with release dates solved optimally within 1000 sec.

	n			
k \ m	10	20	30	40
k=2	0.0	0.7	18.7*	35.7*
m=2	0.0	0.5	8.6*	37.5*
m=3	0.0	1.6	47.5*	52.5*
m=6	0.0	0.1	0.3	20.7*
k=3	0.1	1.9	52.4*	74.2*
m=2	0.1	3.1	39.7*	95.5*
m=3	0.0	2.5	117.4*	150.0*
m=6	0.0	0.2	1.2	6.6*
k=4	0.1	4.0	53.9*	105.3*
m=2	0.2	7.1	85.0*	187.5*
m=3	0.2	4.8	86.1*	200.4*
m=6	0.0	0.3	1.8*	15.9*
k=9	0.5	50.8*	159.1*	95.3*
m=2	0.8	102.7*	296.0*	343.4*
m=3	0.7	50.0*	340.2*	341.6*
m=6	0.1	0.6	20.8*	29.9*

Table 8 Average computing time for parallel unrelated machine instances with release dates solved optimally (s.).*Not all instances were solved within the 1000 s. time limit. +No instance was solved within the time limit.

Well-known special cases: $1|r_j|\sum w_jU_j$, $R|r_j|\sum w_jU_j$ Applied to the problem of minimizing the weighted number of tardy jobs ($K = 1$), the model allows to solve 151 out of the 160 100-job instances tested on a single processor within a time limit of 1000 seconds. On parallel unrelated machines, 467 out of 480 40-job instances (resp. 415 out of 480 50-job instances) are solved optimally within the same time limit. We were not able to find recent papers addressing this specific problem in the literature.

5 Conclusion

Mixed Integer Linear Programming models have been proposed for an original problem of scheduling inspection operations subject to a fixed production schedule. The results obtained using a straightforward application of a standard MILP solver on the general problem as well as some of its special cases support the development of dedicated solving methods based on these models, such as surrogate or Lagrangean relaxation approaches.

Acknowledgements This work was supported by STMicroelectronics and the research project Rousset 2003-2009, financed by the Communauté du Pays d'Aix, Conseil Général des Bouches du Rhône and Conseil Régional Provence Alpes Côte d'Azur.

References

1. Carlier, J., Problèmes d'ordonnancement à contraintes disjonctives, Thèse de troisième cycle, Université de Paris VI (1975)
2. Carlier, J., The one-machine sequencing problem, European Journal of Operational Research, 11, 42-47 (1982)
3. Dauzère-Pérès, S. and Sevaux, M., Using lagrangean relaxation to minimize the weighted number of late jobs on a single machine, Naval Research Logistics, 50, 273-288 (2002)

4. Jackson, J.R., Scheduling a Production Line to Minimize Maximum Tardiness, Research Report 43, Management Science Research Project, University of California, Los Angeles (1955).
5. Jouglet, A., Savourey, D., Carlier, J. and Baptiste, P., Dominance-based heuristics for one-machine total cost scheduling problems, *European Journal of Operational Research*, 184, 879-899 (2008).
6. M'Hallah, R. and Bulfin, R. L., Minimizing the weighted number of tardy jobs on parallel processors, *European Journal of Operational Research*, 160, 471 – 484 (2005).
7. Tanaka, S. and Fujikama, S., An efficient exact algorithm for general single-machine scheduling with machine idle time, In *IEEE International Conference on Automation Science and Engineering*, 371–376 (2008).

Tabu Search Techniques for the Heterogeneous Vehicle Routing Problem with Time Windows and Carrier-Dependent Costs

Sara Ceschia · Andrea Schaerf

Abstract In this work we formalize a new complex variant of the classical vehicle routing problem arising from a real-world application. Our formulation includes a heterogeneous fleet, a multi-day planning horizon, a complex carrier-dependent cost function for the vehicles, and the possibility of leaving orders unscheduled.

We propose a metaheuristic approach based on tabu search and on a complex combination of neighborhood relations. Finally, we perform an experimental analysis to tune and compare different combinations, highlighting the most important features of the algorithm.

1 Introduction

Vehicle routing is one of the most studied problems in optimization (see, e.g., [1]). Many variants of the vehicle routing problem (*VRP*) have been introduced in the literature in the recent years, ranging from multi-depot, to time windows, to mixed fleet, just to name some. Nevertheless, despite the availability of this large set of classified formulation, sometimes the practical problem that companies have to face is still more complex than the standardized version discussed in scientific articles.

This is the case of the problem we have been come across, and thus in this work, we consider a new version of the VRP problem. We decided to work on its exact real-world formulation, without any concession to “judicious simplification”, that would have probably allowed us to borrow results from existing successful solution techniques.

Our formulation, explained in detail in Section 2, includes a heterogeneous fleet, a multi-day planning horizon, a complex carrier-dependent cost function for the vehicles, and the possibility of leaving orders unscheduled.

We propose for this problem a solution technique based on tabu search (*TS*) [2,3], that makes use of a combination of different neighborhood relations. The experimental analysis is carried out on a few real-world instances, and makes use of principled statistical tests to compare different variants and to tune the parameters.

Sara Ceschia and Andrea Schaerf
DIEGM, University of Udine
via delle Scienze 208, I-33100, Udine, Italy
E-mail: {sara.ceschia,schaerf}@uniud.it

The final outcome of the experimental analysis is that the most promising techniques are obtained by a combination of different neighborhood structures.

The paper is organized as follows. In Section 2 we present the problem formulation. In Section 3 we discuss related work. The application of tabu search to the problem is illustrated in Section 4. Section 5 shows the experimental analysis. Finally, in Section 6 we draw some conclusions and discuss future work.

2 Problem Formulation

We present our problem in stages by showing one by one the features (either previously published or original) that are included in the formulation (in Section 2.1). In Section 2.2, we describe in detail the costs associated with the use of the vehicles. Finally, in Section 2.3 we summarize the overall objective function of the problem.

2.1 Features of the Problem

We describe our formulation starting from the basic version of VRP, the *Capacitated VRP*. We have the following entities and constraints:

1. Customer: Each customer requires a supply of a certain amount of goods (*demand*).
2. Fleet: The transportation of goods is performed by using a fleet of vehicles. The vehicles are identical (i.e., they have the same capacity) and located at a single central depot (*home depot*), where they have to return upon complete delivery.
3. Load Limits: The load of each vehicle cannot exceed the vehicle capacity.

The solution of a VRP calls for the determination of a set of *routes*, each performed by a single vehicle that starts and ends at the depot, such that all the requirements of the customers are fulfilled and the global transportation cost is minimized. The travel cost can be either the distance or a different measure of the total expenses of going on a given street (time, tolls, ...).

The first extensions we consider are represented by the so-called *Time Windows* and *Service Times*.

4. Time Windows: Each customer is associated with a time interval. The service of each customer must start within that interval (window). In case of early arrives at the location of a customer, the vehicle is allowed to wait until the service can start.
5. Service Times: Each customer is associated with a service time needed to unload the goods from the vehicle. The vehicle must stop at the customer location for the service time.

The routing cost is equal to the sum of all the times spent along the route: the traveling time, the service time, and the possible waiting time.

The depot also has a time window, that usually includes all the windows of the customers. All vehicles leave at the start time of the depot window, and they must be back before the end of it.

Secondly, we introduce the possibility that the fleet is heterogeneous and limited:

6. Vehicles and Carriers: Each vehicle has its own capacity and belongs to a *carrier*. Each carrier uses a different function to compute the routing costs, as detailed in Section 2.2, depending on the capacity of the vehicle and the length of the route.

7. Limited fleet: The number of available vehicles of each capacity and each carrier is fixed.

In our application, the planning period is not limited to a single day of exercise, but it spans over several days. To this aim, we introduce the following quantities:

8. Planning period: The planning period is composed by a number of days. Thus we have to design a set of routes for each day of a given planning period. Each vehicle performs only one route per day (it must return at the depot at the end of the day). Every day of the period we have the same fleet available.
9. Delivery dates for orders: A customer can make more than one order during the planning period, each with a different demand. For each order, the customer specifies in which days, during the planning period, the goods can be delivered.

The next extension regards the mandatory rests of drivers. In fact, by regulation, drivers must take breaks during their driving duty:

10. Driving rests: Drivers should take breaks of at least 45 minutes after 4 hours and half of consecutive driving. Therefore, in the schedule, we have to include the rest time in every route that is longer than 270 minutes.

Due to site topology and road barriers, there might be limitations on the possibility of using some vehicles to serve certain costumers.

11. Site reachability: There is a compatibility relation between customers and vehicles and every customer has to be served only by compatible vehicles.

Finally, in our case some orders might be left out of the schedule.

12. Exclusion of orders: Orders are split into *mandatory* and *optional*. Optional orders can be excluded from the schedule and consequently they have an associated fixed cost because they are not scheduled.

2.2 Vehicle Cost Functions

Carrier companies have different ways to bill the customer, based on the route and the size of the delivered goods. To this regard, our solver for computing the cost invokes an external code, that in principle could contain any computable function.

However, in the cases we have examined this is restricted to the following four functions:

1. A fixed cost for the vehicle plus a cost per travel unit (Km).
2. A fixed cost for the vehicle plus a cost per load unit (Kg) dependent on the farthest location.
3. A fixed cost for the vehicle plus a cost per travel unit up to a predefined level of load (dependent on the vehicle capacity), a cost per load unit for larger loads.
4. A fixed cost for the vehicle plus a cost per load unit, with different values dependent on the total load and on the farthest location.

In fact, each customer is located in a region and the carrier specifies the load cost coefficient for every region. The coefficient used to compute the cost per load unit is the largest of the route, i.e. the one associated with the region of the farthest customer (with respect to the depot).

2.3 Objective Function

As usual, we split the constraints in hard and soft ones. The first ones must be necessarily satisfied by a solution, whereas the latter ones can be violated and they contribute to the objective function to be minimized.

In our problem, there are three types of hard constraints:

- H1 The capacity of the vehicle cannot be exceeded.
- H2 The return time to the depot cannot be later than a fixed time, that in our case is set to 1 hour after the end of the time window of the depot. In other words, a late return up to 1 hour is penalized but still feasible, a longer one is infeasible.
- H3 The site reachability has to be guaranteed for each pair customer-vehicle.

The objective function is a weighted linear combination of many components that are related to the features described in Section 2.1. The cost components are the following:

- S1 Delivery of an order in a day not included in its delivery days.
- S2 Delivery of an order in a time outside its window.
- S3 Optional orders not delivered.
- S4 Cost of using a vehicle (as described in Section 2.2).

Notice that S1, S2, and S3 are soft constraints, while S4 refers to the cost to be minimized. In addition, we have a component that takes into account the *empty space* in each vehicle:

- S5 The empty space in the vehicles.

This is performed by a non-linear function that penalizes a balanced load on all vehicles in favour of large free spaces in some of them and small in others. This latter case is preferable because large unused spaces can be more easily used for possible unforeseen changes and additional loads.

The weights of the various components are not fixed, but they are set by the operator for each specific instance.

3 Related Work

The Vehicle Routing Problem was first introduced by Dantzig and Ramsey [4] in what they called *The Truck Dispatching Problem*. It was formulated as a branch of the Traveling Salesman Problem (TSP) with multiple vehicles and routes. Subsequently, many other extensions that include time windows, different depots, pick-up and delivery options, heterogeneous fleet and periodic routing have been developed (see the review by Toth and Vigo [5]).

We review here the articles in the literature that deal with those variants of the VRP that are the closest to our problem.

Baldacci *et al* [6] give an overview of approaches to solve the case of a fleet of vehicles characterized by different capacities, the *HVRP* (H for Heterogeneous). Some recent construction heuristics for the HVRP include the ones based on column generation methods [7,8] and on sweep-based algorithm [9].

The first metaheuristic for HVRP was proposed in [10]. Then tabu search approaches for this problem family are developed in [11,12,13]. Ochi *et al* [14] propose

a parallel genetic algorithm in conjunction of scatter search. Tarantilis *et al* [15,16] present a list-based threshold accepting metaheuristic where a randomly selected solution is accepted only if the difference between the cost of the new solution and the cost of the current one is less than a control parameter that is reduced during the search process in a non-monotonic schedule. Liu *et al* [17] develop a record to record travel algorithm and they propose a new set of five test instances that have 200–360 customers.

The HVRP with Time Windows (*HVRPTW*) has been much less studied than the HVRP. The first work is reported by Liu and Shen [18], where they develop a saving-based construction heuristic. They also create three sets of instances to test this new problem variant. More recently, Dell'Amico *et al* [19] proposed an algorithm based on a parallel insertion procedure. Bräysy *et al* [20] presented a solution approach divided in three phases: firstly, a saving-based heuristic creates an initial solution, then a local search procedure focuses on the reduction of the vehicle and finally, the solutions are improved by a simulated annealing metaheuristic.

The VRP with Private fleet and Common carrier (*VRPPC*) describes a situation where, for hypothesis, the total demand exceeds the capacity of the internal fleet, so external transporter is necessary (*common carrier*). The objective is to supply customers, using the private fleet or the common carrier, minimizing the routing costs. VRPPC was first proposed in [21], but, in spite of its practical importance, up to our knowledge it has attracted less research effort. The problem is solved in [22,23,24] always using heuristic approaches.

Our problem is related also to the so-called *Team Orienteering Problem (TOP)*. In the TOP a set of potential customers is available and a profit is collected from the visit to each customer. A fleet of vehicles is available to visit the customers, within a given time limit. The objective is to identify the customers which maximize the total collected profit while satisfying the given time limits for each vehicle. The TOP appeared first under the name *Multiple Tour Maximum Collection Problem* in [25], but the definition of TOP was introduced by Chao *et al* in [26] that proposed a heuristic algorithm. Tabu search approaches have been recently presented by [27,28].

4 Application of Tabu Search

In order to apply tabu search to our VRP problem we have to define several features. We first illustrate the search space and the procedure for computing the initial state. Then, we define the neighborhood structure and the prohibition rules, and finally we describe the set of search components employed and the high-level strategies for combining them.

4.1 Search Space and Initial Solution

The local search paradigm is based on the exploration of a search space composed of all the possible complete assignments of values to the decision variables, possibly including also the infeasible ones.

In this case, first of all a solution defines if the order is scheduled or it is excluded. Then, for each scheduled order, the solution has to specify the day when the order is

delivered, the vehicle, and the position in the corresponding route (the arriving time at the client is deterministically computed given the position).

The initial solution is selected at random. That is, we create a state of the search space that satisfies only the constraints about the delivery day and the site reachability. This is made by assigning each order to a randomly selected feasible day of the planning horizon, then to a random vehicle between those that can reach the customer location and finally to a random position in the selected route. At this first stage, all the orders are scheduled.

4.2 Neighborhood Relations

The neighborhood of a solution is usually implicitly defined by referring to a set of possible moves, which define transitions between solutions. In our problem, we are dealing with the assignment of an order to three kinds of resources: the days, the vehicles and the position in the route.

The first neighborhood, called **Insertion (Ins)**, is defined by simply removing an order from a route and inserting it in another one in a specific position. An order can also be inserted in the list of the unscheduled ones (the position is indifferent) or put back from this list to a route. The list of unscheduled orders is in practice treated as an additional special route, with the main difference that the position is irrelevant.

A move of type **Ins** is identified by 5 attributes $\langle o, or, op, nr, np \rangle$ where o represents an order, or and op the old route and the old position in the old route, and nr and np the new route and new position in the new route, respectively.

The **Inter-route swap (InterSw)** neighborhood, instead, is defined by exchanging an order with another one that is in a different route. A move of this type is identified by 6 attributes $\langle o1, o2, r1, r2, p1, p2 \rangle$, where $o1$ and $o2$ are orders, $r1$ and $r2$ are the routes of $o1$ and $o2$, $p1$ and $p2$ are the positions of the orders in the routes.

Finally, the **Intra-route swap (IntraSw)** neighborhood, is defined by exchanging an order with another one in the same route. A move of this type is identified by 5 attributes $\langle o1, o2, r, p1, p2 \rangle$, where $o1$ and $o2$ are orders, r is the route, $p1$ and $p2$ are the positions of the orders in the route.

4.3 Prohibition Rules

A prohibition rule determines which moves are prohibited by the fact that a given move m is tabu. The prohibition rules are based on the values of the attributes of the two moves, the one in the list and the one under evaluation.

For each neighborhood structure we have tested several prohibition rules of different restrictive level. For the **Ins** neighborhood, assuming that the move $m_t = \langle o_t, or_t, op_t, nr_t, np_t \rangle$ is in the tabu list and the move $m_e = \langle o_e, or_e, op_e, nr_e, np_e \rangle$ is the move to be evaluated, we consider the 6 alternatives shown in Table 1. The last column shows the percentage (on all moves) of the entire neighborhood that a single move prohibits if it happens to appear in the tabu list. In detail, for a given state, the value is calculated as the ratio of number of couple of moves where the first one is the inverse of the other to the total number of couple of moves.

Rule	Condition	Description of tabu moves	Tabu level
PR1	$o_e = o_t \wedge or_e = nr_t$	moves removing the order o_t from the route nr_t	0.098%
PR2	$nr_e = or_t \wedge or_e = nr_t$	moves putting back any order from nr_t to or_t	0.417%
PR3	$o_e = o_t \wedge nr_e = or_t$	moves reinserting the order o_t in the route or_t	1.794%
PR4	$o_e = o_t$	moves involving the same order o_t	1.799%
PR5	$or_e = nr_t$	moves removing any order from the route nr_t	5.054%
PR6	$nr_e = or_t$	moves reinserting any order into the route or_t	7.923%

Table 1 Prohibition rules for TS(Ins)

Similarly, we have tested several prohibition rules also for the two neighborhoods **IntraSw** and **InterSw**. However, for the sake of brevity, we report here only the ones which proved to be the most effective for our instances.

Specifically, for **IntraSw**, if $m_e = \langle o1_e, o2_e, r_e, p1_e, p2_e \rangle$ is the move to be tested and $m_t = \langle o1_t, o2_t, r_t, p1_t, p2_t \rangle$ is in the tabu list, the condition

$$o1_e = o1_t \vee o2_e = o1_t$$

is imposed. It forbids to make a move where any order of m_e is equal to the first order of any move m_t belonging to the tabu list.

For **InterSw**, if $m_e = \langle o1_e, o2_e, r1_e, r2_e, p1_e, p2_e \rangle$ is the move to be tested and $m_t = \langle o1_t, o2_t, r_t, p1_t, p2_t \rangle$ is the move in the tabu list, then

$$o1_e = o1_t \vee o2_e = o1_t$$

is the condition imposed; so that, it is forbidden to make a move that involves the same two routes of the tabu one.

4.4 Tabu Dynamics

We make use of the Robust Tabu Search scheme, as called in [3]. In this scheme the tabu tenure is *variable*, consequently each performed move remains in the list for a number of iterations randomly selected between a minimum and a maximum value. In detail, we set two values t and δt and we assign to each accepted move a tabu tenure randomly selected between $t - \delta t$ and $t + \delta t$. In all our experiments δt is set to 2 (based on preliminary runs), whereas t is subject to tuning.

4.5 Search Techniques

Based on the three neighborhood relations, we have three basic Tabu Search techniques, that we call **TS(Ins)**, **TS(IntraSw)**, **TS(InterSw)**.

Notice however that under the **IntraSw** and **InterSw** neighborhood structures the search space is not connected. In fact, for both it is not possible to change the number of orders in a route, thus there is no trajectory that goes from a state with a given number of orders in a route to a state with a different one. Therefore, for our problem **TS(IntraSw)** and **TS(InterSw)** are not effective as search techniques by themselves, but they must be used in combination with **TS(Ins)**.

We use a sequential solving strategy for combining tabu search algorithms based on different neighborhood functions, as proposed (among others) in [29]: given an initial

Name	#O	#C	#V	#VT	#R	#D
PAC1-A	139	44	7	4	19	4
PAC1-B	166	56	7	4	22	4

Table 2 Features of the instances

state and a set of algorithms, its search makes *circularly* a run of each algorithm, always starting from the best solution found by the previous one. The overall process stops either when a full round of all of them does not find an improvement or the time granted is elapsed. Each single technique stops when it does not improve the current best solution for a given number of iterations (*stagnation*).

We thus identify five strategies (denoted by the $\triangleright$ symbol in [29]) resulting of the combination of the basic neighborhood structures: $\text{TS}(\text{Ins})$, $\text{TS}(\text{Ins})\triangleright\text{TS}(\text{IntraSw})$, $\text{TS}(\text{Ins})\triangleright\text{TS}(\text{InterSw})$, $\text{TS}(\text{Ins})\triangleright\text{TS}(\text{IntraSw})\triangleright\text{TS}(\text{InterSw})$, and $\text{TS}(\text{Ins})\triangleright\text{TS}(\text{InterSw})\triangleright\text{TS}(\text{IntraSw})$.

We also consider solvers that use the union of many neighborhood: the algorithm based on this compound neighborhood (denoted by the $\oplus$ symbol in [29]) selects, at each iteration, a move belonging to any of the neighborhoods as part of the union. We compare for example the $\text{TS}(\text{Ins})\triangleright\text{TS}(\text{InterSw})$ solver with the $\text{TS}(\text{Ins}\oplus\text{InterSw})$ one, that makes use of the union of Ins and InterSw . We do not report the results in detail, because preliminary experiments show that the union regularly gives worse results.

5 Experimental Analysis

In this section, we first introduce the benchmark instances and the general settings of our analysis, and then we move to the experimental results. We analyze our techniques in stages, starting from the simpler algorithm which uses one single neighborhood, then moving to the more complex ones.

5.1 Benchmark Instances

We have four instances coming from real cases: two of them completely independent, the others with the same data but a different partition of orders into mandatory and optional. For each instance, we tested three different realistic configurations of the weights of the cost components. The configurations of the weights make the solutions quite different from each other, thus resulting in 12 independent instances.

Table 2 summarizes the ranges of the size of the instances in terms of number of orders ($\#O$), number of clients ($\#C$), number of vehicles ($\#V$), number of vehicle types ($\#VT$), number of regions ($\#R$) and number of days of the planning period ($\#D$).

5.2 General Settings and Implementation

All the algorithms have been implemented in the C++ language, exploiting the EASYLOCAL++ framework [30]. The experiments were performed on an Intel QuadCore PC (64 bit) running Debian Linux 4.0, the software is compiled using the GNU C++ compiler (v. 4.1.2).

In order to compare different combinations in a fair way, we grant to all of them the same running time for each run. In particular, in all experiments we grant 100

Prohibition Rule	Tabu Length	Avg. Cost	p-value
PR5	15	230345.72	—
PR6	35	233419.30	$3.29 \cdot 10^{-13}$
PR4	35	232501.85	$7.81 \cdot 10^{-29}$
PR1	35	232661.77	$1.49 \cdot 10^{-29}$
PR3	45	237109.48	$6.40 \cdot 10^{-52}$
PR2	40	242538.63	$9.95 \cdot 10^{-64}$

Table 3 Comparison of prohibition rules for TS(Ins)

seconds for each trial, and we run 30 trials for each configuration of the parameters. The random seed is used as *blocking factor*, so that all configurations have the same 30 random seeds, and thus they start from the same initial solution.

For each individual TS(·) component of the search strategy the maximum number of iterations from the last improvement is set to 500. The full strategy stops only based on the timeout, independently of the improvements obtained in the last rounds.

To compare the results we use the *Student's t-test*, provided that the underlying distributions can be assumed to be normal and independent [31].

If the calculated p-value is below the threshold chosen for statistical significance (typically 0.05), then the null hypothesis which usually states that the two groups do not differ, is rejected in favor of an alternative hypothesis, which states that an algorithm is superior to another on the proposed instances.

The t-test has two different versions depending on whether the two samples are:

- unpaired, independent of each other, or
- paired, so that each member of one sample has a unique relationship with a particular member of the other sample.

In our experiments, we use the paired version by associating each run of one algorithm with the one of the other algorithm on the same instance and having the same random seed (thus the same initial solution).

5.3 Results on Prohibition Rules for the Insertion Neighborhood

The first set of experiments focuses on the **Insertion** neighborhood, and compares the different prohibition rules of TS(Ins).

Obviously, for each prohibition rule the best tabu length can be different. Therefore we have to tune the length for each rule independently, and then compare the prohibition rules among themselves, each with its best setting.

Table 3 shows the comparison between the best settings of each rule. It presents, for each rule, the selected length value and the average cost, in ascending order. The last column shows the p-value of the comparison between the best configuration (PR5, first line) and the current one. We remember that a small p-value (lower than 0.05, separated by an horizontal line in the tables) means that the two distribution are significantly different and that one configuration is superior to the other on these instances.

The table shows that the p-values are extremely small, the the prohibition rule PR5 is clearly superior to all the others. It also shows that for the selected rule PR5 the best length is considerably smaller than for all the others. In the subsequent experiments we use only this rule.

IntraSw			InterSw		
Tabu Length	Avg. Cost	p-value	Tabu Length	Avg. Cost	p-value
6	231659.8	—	30	230210.94	—
4	231700.04	0.50	20	230256.03	0.66
8	231719.22	0.32	25	230333.95	0.05
10	231742.35	0.16	35	230482.98	0.04

Table 4 Comparison of tabu length of TS(IntraSw) and TS(InterSw)

Solver	Avg. Cost	p-value
TS(Ins)▷ TS(InterSw)	230430.12	—
TS(Ins)▷ TS(IntraSw)▷ TS(InterSw)	230456.04	0.67
TS(Ins)▷ TS(InterSw)▷ TS(IntraSw)	230504.91	0.16
TS(Ins)▷ TS(IntraSw)	231109.04	$7.55 \cdot 10^{-13}$
TS(Ins)	231146.45	$5.525 \cdot 10^{-14}$

Table 5 Comparison of composite solvers

5.4 Results on Inter-route swap and Intra-route swap Neighborhoods

As already noted, TS(InterSw) and TS(IntraSw) cannot be used alone because the search space is not connected under their neighborhood relation, thus only a subset of all possible solution would be explored. Consequently, in order to tune the parameters of TS(InterSw) and TS(IntraSw) we make use of the strategies TS(Ins)▷TS(InterSw) and TS(Ins)▷TS(IntraSw), respectively.

We report in Table 4 the tuning of the tabu length of TS(IntraSw) and TS(InterSw), keeping the parameters of TS(Ins) as set in the previous experiment.

Table 4 shows that TS(IntraSw) is not much affected to tabu length variations (p-values are high). Anyway the best configuration has a short tabu list. The outcome is not distinct also for TS(InterSw), in fact the most fitting tabu length is 30, but 20 wins over 25.

5.5 Results on Composite Solvers

Our last experiment regards the comparison of the composite solvers. For each component of the solvers, the parameters are set to the value obtained in the previous experiments.

Table 5 reports the results, ordered as usual by the p-values w.r.t. to the best configuration.

The outcome is that the best solver is TS(Ins)▷TS(InterSw), but there are other two combinations, namely TS(Ins)▷TS(IntraSw)▷TS(InterSw) and TS(Ins)▷TS(InterSw)▷TS(IntraSw), that are not statistically distinguishable.

6 Conclusions and Future Work

We have proposed a hybrid metaheuristic approach based on tabu search for a highly complex version of the classical vehicle routing problem arising from a real world situation.

The experimental analysis shows that the best results are obtained by a combination of **Ins** and **InterSw** neighborhood, whereas the use of the **IntraSw** one is not useful in our cases.

For the future we plan to test the use of other neighborhood relations and other techniques and to compare them in a principled way. In addition, we plan to investigate the relative importance of the various cost components and the relation of their weights with the tuning of the parameters of the solver.

Acknowledgements We thank Luca Di Gaspero for helpful comments on the paper. We are also grateful to BeanTech s.r.l. for bringing this problem to our attention, providing to us the instances, and supporting Sara Ceschia with a grant for this work.

References

1. Toth, P., Vigo, D. (eds.): The Vehicle Routing Problem. Monographs on Discrete Mathematics and Applications. S.I.A.M., Philadelphia, PA (USA) (2002)
2. Glover, F., Laguna, M.: Tabu search. Kluwer Academic Publishers (1997)
3. Hoos, H.H., Stützle, T.: Stochastic Local Search – Foundations and Applications. Morgan Kaufmann Publishers, San Francisco, CA (USA) (2005)
4. Datzig, G., Ramser, J.: The truck dispatching problem. *Management Science* **6**(1), 80–91 (1959)
5. Toth, P., Vigo, D.: An overview of vehicle routing problems. In: The vehicle routing problem, pp. 1–26. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2001)
6. Baldacci, R., Battara, M., Vigo, D.: Routing a heterogeneous fleet of vehicles. Technical report deis or. inge 2007/1, DEIS, University Bologna, via Venezia 52, 47023 Cesena, Italy (2007)
7. Taillard, E.: A heuristic column generation method for the heterogeneous fleet vrp. *RAIRO Recherche Oprationnelle* **33**(1), 1–14 (1999)
8. Choi, E., Tcha, D.W.: A column generation approach to the heterogeneous fleet vehicle routing problem. *Comput. Oper. Res.* **34**(7), 2080–2095 (2007)
9. Renaud, J., Boctor, F.F.: A sweep-based algorithm for the fleet size and mix vehicle routing problem. *European Journal of Operational Research* **140**, 618–628 (2002)
10. Semet, F., Taillard, E.: Solving real-life vehicle routing problems efficiently using tabu search. *Annals of Operations Research* **41**, 469–488 (1993)
11. Osman, I.H., Salhi, S.: Local search strategies for the vehicle fleet mix problem. *Modern Heuristic Search Methods* (1996)
12. Gendreau, M., Laporte, G., Musaraganyi, C., Taillard, E.D.: A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & Operations Research* **26**(12), 1153–1173 (1999)
13. Wassan, N.A., Osman, I.H.: Tabu search variants for the mix fleet vehicle routing problem. *Journal of the Operational Research Society* **53**(7), 768–782 (2002)
14. Ochi, L.S., Vianna, D.S., Drummond, L.M.A., Victor, A.O.: A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. *Parallel And Distributed Processing* **1388**, 216–224 (1998)
15. Tarantilis, C.D., Kiranoudis, C.T., Vassiliadis, V.S.: A list based threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. *Journal of the Operational Research Society* **54**(1), 65–71 (2003)
16. Tarantilis, C.D., Kiranoudis, C.T., Vassiliadis, V.S.: A threshold accepting metaheuristic for the heterogeneous fixed fleet vehicle routing problem. *European Journal of Operational Research* **152**(1), 148–158 (2004)
17. Li, F., Golden, B., Wasil, E.: A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. *Comput. Oper. Res.* **34**(9), 2734–2742 (2007)
18. Liu, F.H., Shen, S.Y.: The fleet size and mix vehicle routing problem with time windows. *Journal of the Operational Research Society* **50**(7), 721–732 (1999)

19. Dell'Amico, M., Monaci, M., Pagani, C., Vigo, D.: Heuristic approaches for the fleet size and mix vehicle routing problem with time windows. Tech. rep., DISMI, University of Modena and Reggio Emilia, Italy (2006)
20. Bräysy, O., Dullaert, W., Hasle, G., Mester, D., Gendreau, M.: An effective multi-restart deterministic annealing metaheuristic for the fleet size and mix vehicle routing problem with time windows. *Transportation Science* (2006)
21. Volgenant, T., Jonker, R.: On some generalizations of the travelling-salesman problem. *Journal of the Operational Research Society* **38**(11), 1073–1079 (1987)
22. Diaby, M., Ramesh, R.: The distribution problem with carrier service: a dual based approach. *ORSA Journal on Computing* **7**(1), 24–35 (1995)
23. Bolduc, M., Renaud, J., Boctor, F.: A heuristic for the routing and carrier selection problem (2005). Working paper
24. Chu, C.W.: A heuristic algorithm for the truckload and less-than-truckload problem. *European Journal of Operational Research* **127**(3), 657–667 (2005)
25. Butt, S.E., Cavalier, T.M.: A heuristic for the multiple tour maximum collection problem. *Comput. Oper. Res.* **21**(1), 101–111 (1994)
26. Chao, I.M., Golden, B.L., Wasil, E.A.: The team orienteering problem. *European Journal of Operational Research* **88**(3), 464 – 474 (1996)
27. Tang, H., Miller-Hooks, E.: A tabu search heuristic for the team orienteering problem. *Comput. Oper. Res.* **32**(6), 1379–1407 (2005)
28. Archetti, C., Hertz, A., Speranza, M.: Metaheuristics for the team orienteering problem. *JH* **3**, 49–76 (2007)
29. Di Gaspero, L., Schaerf, A.: Neighborhood portfolio approach for local search applied to timetabling problems. *Journal of Mathematical Modeling and Algorithms* **5**(1), 65–89 (2006)
30. Di Gaspero, L., Schaerf, A.: EASYLOCAL++: An object-oriented framework for flexible design of local search algorithms. *Software—Practice and Experience* **33**(8), 733–765 (2003)
31. Venables, W.N., Ripley, B.D.: *Modern Applied Statistics with S*. Springer (2003)

A Water Flow Algorithm for Flexible Flow Shop Scheduling with Limited Intermediate Buffers

Tran Trung Hieu, Ng Kien Ming

Abstract: In this paper, we propose a water flow algorithm for solving flexible flow shop scheduling problem with limited intermediate buffers, which is a NP-hard problem encountered in industry. The proposed algorithm is inspired by the hydrological cycle in meteorology and the erosion phenomenon in nature. In the algorithm, we combine the amount of precipitation and its falling force to form a flexible erosion capability, which helps the erosion process focus on exploiting promising regions strongly. We also integrate a seed position obtained by a constructive algorithm into the initial population to increase the performance of the algorithm. Some benchmark instances of the scheduling problem are used to evaluate our proposed algorithm. The computational results and comparisons demonstrate the efficacy of the algorithm. In addition, an industrial example is also introduced to illustrate the algorithm's efficiency when solving practical problems.

Keywords: Water flow algorithm, metaheuristics, flexible flow shop, limited buffers.

1 Introduction

The flexible flow shop scheduling problem (FFSP) with limited intermediate buffers consists of a set of jobs processed through several consecutive operation stages with parallel identical machines in each stage and there are limited intermediate buffers between consecutive stages. This type of problem arose significantly in both traditional and modern manufacturing systems, such as the just-in-time production, the continuous casting-hot charge rolling production in steel industry, or the printed circuit board assembly line in electronics industry.

Although many researchers have focused on developing optimization techniques for solving various types of flow shop scheduling problem, only a few of them dealt with the FFSP with limited buffers [2]-[7]. Through these works, it can be seen that the approach of solving the FFSP with limited buffers by metaheuristic algorithms has become popular and

Tran Trung Hieu

Department of Industrial and Systems Engineering, National University of Singapore,
10 Kent Ridge Crescent, Singapore 119260
E-mail: g0600324@nus.edu.sg

Ng Kien Ming

Department of Industrial and Systems Engineering, National University of Singapore,
10 Kent Ridge Crescent, Singapore 119260
E-mail: kienming@nus.edu.sg

received more attention by researchers. However, the quality of solutions obtained by these algorithms may not be good, especially for problems with large size.

In this paper, a novel water flow algorithm (WFA) for solving the FFSP with limited buffers is proposed. The WFA is based on simulating the hydrological cycle in meteorology and the erosion phenomenon in nature. It is an efficient metaheuristic algorithm which combines solution exploration and exploitation capabilities. To evaluate the performance of the WFA, we have tested it on some FFSP with limited buffers from the literature. In addition, a problem of this type encountered in the maltose syrup production industry is also introduced and used to evaluate the efficiency of the WFA in solving practical applications.

The remaining content of this paper is organized as follows. The FFSP with limited buffers will be introduced in the next section. In Section 3, the proposed WFA for solving the FFSP with limited buffers is described in detail, while Section 4 describes the maltose syrup production problem. Computational experiments and comparisons based on some benchmark problems of the FFSP with limited buffers and the maltose syrup production problem are presented in Section 5. Finally, some conclusions and future work are mentioned in Section 6.

2 FFSP with limited intermediate buffers

In general, the FFSP with limited intermediate buffers is a typical NP-hard combinatorial optimization problem that may be formulated as follows. A set of n jobs is processed on s consecutive production stages with m_l parallel identical machines in each stage l ($l = 1, \dots, s$) and there are limited buffers between these consecutive stages (refer to Figure 1). In this problem, jobs have to be processed successively through all s stages. One machine in each stage can only process one job at a time and each job can only be processed on at most one machine in each stage at the same time. In addition, each job is processed without preemption on one machine in each stage. Moreover, a job can skip one or more stages but is unable to go back to a previous stage. Here, only processing time, weight, and due date of jobs are considered in this problem, while other elements such as breakdown time of machines and set-up time of jobs are ignored in this model.

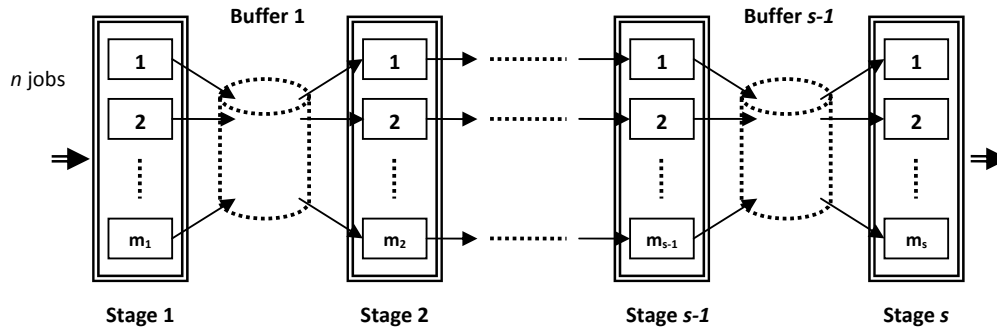


Figure 1: The schematic of FFSP with limited intermediate buffers.

In this paper, we consider the case of no available buffer space for completed jobs between stages. It means that after job j is finished by machine i in stage l , if there is no idle machine in the subsequent stage $l+1$, then job j must wait on machine i in stage l until there is at least an idle machine in stage $l+1$ to start processing job j in this stage.

The common performance measures in the FFSP with no available buffer space include minimizing the completion time of jobs or makespan (C_{max}), minimizing total weighted flow time of jobs, and minimizing total weighted tardiness time of jobs. The aim of these minimization objectives is to help achieve a high throughput for production.

3 WFA for the FFSP with limited buffers

In this section, we will present the operational mechanism of the WFA for solving the FFSP with limited buffers. The WFA is a meteorologically inspired metaheuristic algorithm that mimics the hydrological cycle in meteorology and the erosion phenomenon in nature, which simulates a balance between solution exploration and exploitation capabilities in an optimization algorithm respectively. Since the proposed algorithm is built on simulating the spreading of raindrops in hydrological cycle into many places on the ground, on the essence of water flow always moving from higher positions to lower positions, and on the erosion capability of water flow to the ground, we use some metaphors related to water flow in the WFA. This will help us image more easily about the correlation between the elements in the hydrological cycle and the components in optimization. In addition, it will also be useful for establishing a common basis to develop the algorithms belonging to this approach in the future. In details, the proposed algorithm can be summarized as follows:

Firstly, a cloud representing an iteration will randomly generate a set of drops of water (*DOW*) into some positions representing solutions in the optimization problem on the ground. Next, due to gravity force of Earth represented by a heuristic algorithm, *DOW* will automatically move to local optimal positions. They will be held at these positions until the erosion condition is satisfied to perform erosion process. Then, depending on the amount of precipitation and the falling force of precipitation or soil hardness at the local optimal positions, the erosion process will help *DOW* overcome the local optimal positions to find better local optimal or global positions. A flow chart of the WFA for solving the FFSP with limited buffers is shown in Figure 2. The details of the erosion condition, erosion capability, erosion process, or other operations of the WFA for the FFSP with limited buffers are described in the following subsections.

3.1 Encoding scheme

In the FFSP with limited buffers, we can consider the job permutations and their objective values such as makespan, total weighted flow time of jobs, and total weighted tardiness time of jobs as the longitude, latitude, and altitude information in the position of *DOW* on the ground. In particular, the permutations are the vectors of size n jobs which represent the sequence of the given set of jobs performed in the first stage. For a complete schedule associated with the given permutation, we use the procedure proposed by [6]. Based on the complete schedule, the objective values will be determined. Hence, to simplify in the computational procedure of the WFA for the FFSP with limited buffers, we incorporate the procedure to construct the complete schedule into determining the objective values. With the simple and appropriate solution representation, the FFSP with limited buffers becomes less complex since the search space of the problem is limited in a set of the possible permutations of n jobs. Then, we do not need to pay attention to the number of machines at each stage.

3.2 The best positions list, un-eroded positions list and eroded positions list

Three sets of memories of the WFA to support the search for global optimal positions are used in the FFSP with limited buffer. Firstly, the best positions list, called P_0 -list, saves the job permutations with minimum makespan, total minimum weighted flow time of jobs, or total minimum weighted tardiness time of jobs. Secondly, the un-eroded list, called UE-list, is used to save local optimal positions which have not eroded because of not satisfying the erosion condition. The feature of the second list is to record the potential promising positions for erosion process afterward. Thirdly, the eroded list, called E-list, is used to save eroded local optimal positions. Among these lists, E-list plays an important role in preventing clouds regenerating *DOW* to the eroded positions. It would help to improve the computation time needed by the algorithm.

3.3 Exploration phase

In the first cloud of this phase, a seed job permutation obtained by flexible flow line loading method (FFLL), which is an efficient constructive algorithm for flexible manufacturing systems with cyclic paths [1], is generated. Then, a set of job permutations is obtained by an insertion scheme and is spread around the seed job permutation to be good job permutations for later erosion process. Here, the insertion scheme is performed by removing a job from its present position and inserting it in a different position, and then shifting the position of jobs between these two positions by a unit accordingly.

At next clouds, the FFLL will not be used anymore. The WFA will only use randomly generated job permutations with population size *MaxPop*. For all clouds, after generating a population of job permutations for *DOW*, a steepest ascent hill sliding algorithm is used to search for local optimal positions from these starting positions. In the algorithm, a perturbation scheme based on a 2-opt algorithm is used to identify the solution neighborhood. In general, the exploration phase in the WFA will result in a set of local optimal job permutations. They will be updated in the UE-list to be considered for performing the erosion process in the next exploitation phase.

3.4 Exploitation phase

➤ Erosion condition and capability

In the WFA for the FFSP with limited buffers, we perform the erosion process based on the amount of precipitation. If the amount of precipitation at some local optimal job permutation increases up to *MinEro*, the minimum number of *DOW* allowed to start erosion process, the erosion process will happen at the local optimal job permutation.

In the current WFA, we consider the capability of erosion process based on two main factors, i.e., the amount of precipitation and its falling force or soil hardness. In the FFSP with limited buffers, the amount of precipitation is represented by the number of *DOW* at the eroding local optimal job permutation, while its falling force or soil hardness is represented by the objective function value at the job permutation. We assume that clouds are at the same altitude, and the falling force of precipitation to lower local optimal positions will cause erosion more easily than that of higher ones. In other words, the erosion capability will become stronger to local optimal job permutations with larger amount of precipitation and lower objective function values. It creates a flexible operation scheme for erosion capability of *DOW* in the algorithm and helps the erosion process focus on exploiting real promising regions strongly while ignoring exhausted regions. In particular, the relationship function between these two factors and the control parameter of erosion capability *MaxUIE*, the maximum number of iterations at which the erosion process will move to the next erosion direction if the solution is not improved, is shown as follows:

$$MaxUIE = f(Er[P_L(x, y, z)]; z) = \varphi_1 \times Er[P_L(x, y, z)] + \varphi_2^{LB/z}$$

where φ_1 and φ_2 are parameters representing the effect of precipitation and its falling force respectively. Also, $Er[P_L(x, y, z)]$ is the number of *DOW* in the local optimal position $P_L(x, y, z)$, *LB* is a known lower bound or known optimal solution, and *z* is the objective function value at the eroding local optimal position. Based on preliminary computational experiments, the values for φ_1 and φ_2 are set to 2 and 3 respectively when the WFA is applied to solve the FFSP with limited buffers.

➤ Erosion process

When the erosion condition is satisfied, the erosion process will happen. The erosion capability in this process at local optimal job permutations depends on the above equation. The strategy of erosion process is also based on a topological parameter and block concept. In the FFSP, the topological parameter Δd representing the geographical shape of surface is calculated as the

difference between the objective function value of the local optimum job permutation and that of its neighboring job permutation. The lowest Δd will first be chosen as the erosion direction. If the erosion process for that direction does not improve after $MaxUIE$ iterations, we say that the direction is blocked. It means that water flow cannot move in that direction and searching in that direction will stop. This will be followed by backtracking in which we will restart the search from considering the local optimal job permutation but with another direction. The erosion process will then be carried out for the direction with the next smallest Δd . If all directions for considering the local optimal position are blocked, we call that job permutation fully blocked and move it into the E-list and we do not consider the job permutation in the next clouds anymore. Otherwise, during the process of considering the erosion direction, if there is a direction where there is any improvement when compared with the eroding local optimal job permutation, we will choose that direction to erode permanently for considering the local optimal job permutation. Here, the improvement means that the erosion process finds a new local optimal job permutation whose objective function value is smaller than that of the eroding local optimal job permutation. This new local optimal job permutation will be updated in the UE-list to continue performing the erosion process.

The entire exploration and exploitation process of the WFA for the FFSP with limited buffers will terminate when the maximum number of allowed clouds ($MaxCloud$) is generated.



Figure 2: Flow chart of the improvement WFA for the FFSP with limited buffers.

Even after encoding the solution representation under the permutation vector to reduce the search space of this problem, its combinatorial boom still exists significantly. Hence, it is said that the FFSP with limited buffers is a very difficult problem to solve. However, with the various features shown above such as there the lists, the neighborhood structure, the exploration strategy, and the exploitation strategy representing the erosion process, the WFA may solve this problem efficiently.

4 Example of the FFSP with limited buffers in the maltose syrup production

In developing countries, sugar and candy production industries are of great importance. The main raw material used for such industries is maltose syrup and its production is thus playing a key role in these industries. The maltose syrup production can be modeled as a FFSP with no available buffer space for completed jobs between stages. The problem consists of nine jobs involving nine types of maltose syrups and they are processed through six consecutive stages representing six phases in the maltose syrup production process (see Figure 3). The number of parallel identical machines in each stage, processing time of the jobs on these stages, weight of the jobs, and due date of the jobs are shown in Table 1.

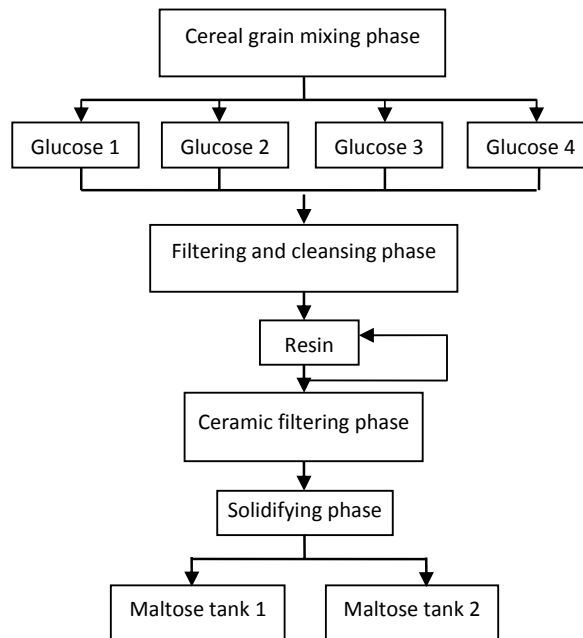


Figure 3: Maltose syrup production process.

To model the maltose syrup production process exactly, we also need to consider some properties in the formulation. Maltose syrup production process is a continuous process which is connected together through pipes and tanks. Here, the jobs processed at the various stages not only depend on the busy/idle state of right consecutive machines as in the standard FFSP with limited buffers, but also depend on the status of other machines. Thus, jobs cannot go to stage 1 until there is at least one idle machine in stage 2 even if the machine in stage 1 is idle. Moreover, since the product is a liquid and stages 2 and 3 are connected directly, the completion time of the jobs in stages 2 and 3 is the same. As an example, Figure 4 illustrates the FFSP with 3 stages (S1-S3) in which there is a machine in stage 1 and stage 3, and 3 machines in stage 2 (M1-M3). 5 jobs (J1-J5) will be processed through these stages in the

order of J1 to J5. J4 can usually be processed at time t_2 for the standard FFSP with no available buffer space. However, due to the special property in maltose syrup production, J4 can only be processed from time t_4 when there is an idle M1 in stage S2. As such, we can consider this model as a FFSP with controlled and limited buffers.

Table 1: Problem data for maltose syrup production

Stage	No. of machines	Processing time of jobs (hours)								
		Job 1	Job 2	Job 3	Job 4	Job 5	Job 6	Job 7	Job 8	Job 9
1	1	6	6	6	6	6	6	6	8	8
2	4	49	49	49	49	49	23	23	11	11
3	1	7	7	7	7	7	7	7	7	7
4	1	2	2	2	2	2	2	2	2	2
5	1	7	5	5	5	5	5	5	7	5
6	1	1	1	1	1	1	1	1	1	1
Due date		120	87	87	172	172	72	72	120	72
Weight		0.10	0.10	0.10	0.05	0.05	0.20	0.20	0.05	0.15

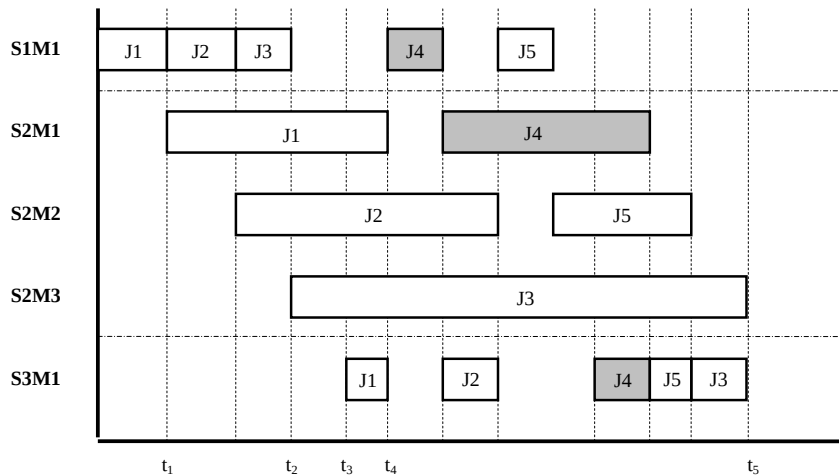


Figure 4: Illustration of FFSP with controlled and limited buffers in maltose syrup production.

5 Computational experiments and comparisons

In this section, we present the results of computational experiments carried out on the benchmark problems of Wittrock [7], Wardono and Fathi [6] and an instance of maltose syrup production mentioned in the previous section. These experiments are used to evaluate the performance of the WFA for the FFSP with limited buffers. The best known upper bounds of these benchmark problems obtained by the algorithm in [6], the best performing algorithm for the benchmark problems used to test here so far, were used to compare with the results obtained by the WFA. In particular, the benchmark problems for the FFSP with limited buffers used to test the performance of the WFA are type I and type III data sets which Wardono and Fathi used to evaluate their algorithm. In fact, the type III data set is the benchmark instances of Wittrock shown in Table 2 with some slight modifications, i.e., the transport time (one minute) of the Wittrock instances is added to the processing time of the jobs at stages 2 and 3, respectively, when applicable [6]. In these instances, the number of machines at stages 1, 2, and 3 are 2, 3, and 3 respectively. There is no buffer between stages.

Table 2: Problem data of Wittrock instances [7]

Job type	Processing time (min)			Production requirement of each instance					
	Stage 1	Stage 2	Stage 3	1	2	3	4	5	6
A	39	11	14	12	-	-	-	-	-
B	13	28	54	1	-	-	-	-	-
C	22	56	60	26	-	-	14	23	20
D	234	39	0	-	-	-	2	-	-
E	39	25	80	-	6	7	4	-	1
F	13	70	54	-	14	20	16	-	-
G	143	66	0	1	4	-	-	3-	1
H	0	28	14	7	-	-	-	-	-
I	26	39	74	-	6	-	-	-	5
J	18	59	34	-	4	-	-	-	-
K	22	70	40	4	-	-	-	-	-
L	13	70	54	-	4	-	-	-	-
M	61	46	34	-	-	11	-	14	3
Total number of jobs				51	38	38	36	40	30

5.1 Platform and parameters

The WFA has been coded using Visual Basic 6.0, and all experiments were performed on an Intel 1.60 GHz CPU with 1.5 GB of RAM. The choice of reasonable parameters for the WFA was determined by design-of-experiment methods and the values are summarized in Table 3. With these parameter sets, 10 independent replicates were used for each instance and the best results obtained were used to evaluate and compare the performance of WFA.

Table 3: Parameter sets for benchmarks

Benchmarks	The value of parameters		
	MaxCloud	MaxPop	MinEro
Modified Wittrock			
Instance 1	10	20	3
Instance 2, 3, 5	10	15	3
Instance 4, 6	10	10	4
Wardono & Fathi	5	10	5
A case study	5	10	5

5.2 Performance measures

For comparison of objective values, we have used the following average relative percentage increase in objective value:

$$\bar{\Delta} = \frac{\sum_{i=1}^R \left(\frac{Heuristic_{sol(i)} - Opt_{sol}}{Opt_{sol}} \right)}{R} \times 100$$

where $Heuristic_{sol(i)}$ and Opt_{sol} denote the objective function value obtained from the proposed WFA for the i th replicate and the best known value or lower bound value in the literature respectively, and R denotes the number of replicates. If we only use the replicate with the best objective value to compute the above $\bar{\Delta}$, the relative percentage increase in objective value will be denoted by $\bar{\Delta}_{Best}$.

When comparing the results obtained by [6] for the benchmark problems of the modified Wittrock, and Wardono and Fathi, we use $\bar{\Delta}_{Best}$ and only the criterion of makespan minimization, while for the maltose syrup production problem, we use $\bar{\Delta}$ with 10 replicates and the objective functions in Section 2.

For a fair comparison of CPU time among the algorithms, we use a quantity called *ratio*, which is the ratio of CPU time for solving an instance in consideration to CPU time for solving the instance with the smallest size.

5.3 Computational results

The comparison results between the WFA and the Tabu search of Wardono and Fathi (TS-H1) [6] for the FFSP with limited buffers are shown in Table 4. Based on the results in this table, it can be seen that the WFA outperforms TS-H1 both in solution quality and CPU time ratio for all the tested instances. The average relative percentage increase in objective value of the WFA is only 6.15%, while it is 7.1% for TS-H1. When solving instances with different sizes, it is also observed that the CPU time of the WFA only increases slightly when compared to TS-H1 as instance size increases. Moreover, for the instance belonging to type I data set of Wardono and Fathi, the WFA is able to obtain the optimal solution as in TS-H1.

Table 4: Comparison results between WFA and TS-H1

Problem	Instance	LB	WFA			TS-H1		
			C_{max}	$\bar{\Delta}_{Best}$	<i>ratio</i>	C_{max}	$\bar{\Delta}_{Best}$	<i>ratio</i>
Wardono & Fathi	1	134.0	134	0.00		134	0.00	
Modified Wittrock	1	746.3	811	8.67	5.61	822	10.14	6.47
	2	758.0	826	8.97	2.93	839	10.69	3.29
	3	758.7	815	7.42	2.26	822	8.35	2.29
	4	755.3	820	8.57	1.26	825	9.22	4.30
	5	961.5	971	0.99	2.39	974	1.30	2.69
	6	666.7	682	2.29	1.00	686	2.90	1.00
Average				6.15			7.10	

In addition, we also compared the results of the proposed WFA for the maltose syrup production problem with the existing practice, whose solution is made by the senior planners' experience, in Table 5, and with the optimal values obtained by the enumeration method. These results show that the solutions obtained by the WFA are much better than the existing solutions in practice. Besides that, the WFA is able to obtain the optimal solutions with a significantly smaller computational time than the enumeration method.

Table 5: Computational results of WFA for the case study

Objectives	Enumeration method		WFA		Existing solution	
	Optimal values	Time (s)	$\bar{\Delta}$	Time (s)	Values	Time (s)
Makespan	139.00	130	0.00	1.4	140.00	
Total weighted flow time	75.95	106	0.00	1.8	110.30	1,800
Total weighted tardiness	0.10	105	0.00	1.7	34.20	

Through these experimental results, we conclude that the WFA is an efficient metaheuristic algorithm for the FFSP with no available buffer space.

6 Conclusions and future work

In this paper, a water flow algorithm that mimics the operation of water flow in hydrological cycle and erosion process in nature is proposed. It involves a constructive algorithm for generating a seed position in the initial population and a relationship function between the amount of precipitation and its falling force with erosion capability to create a flexible operation scheme for erosion capability in the erosion process. This helps the erosion process focus on exploiting promising regions strongly. We show how this algorithm can be applied to solve the FFSP with limited buffers. Computational experiments and comparisons were carried out to show the performance of the proposed algorithm and the results show that it is a promising algorithm not only for solving benchmark problems in the literature but also for solving practical problems.

As the proposed algorithm is applicable to other optimization problems by adapting the components of the algorithm appropriately, one possible future research work is in applying the algorithm to other challenging optimization problems, such as the quadratic assignment problem or vehicle routing problem. Also, certain elements or factors in nature that are related to water flow, such as evaporation and percolation, could possibly be integrated into the WFA to improve its performance.

References

- [1] M. Pinedo, Planning and scheduling in manufacturing and services, pp. 127-131. New York, NY: Springer (2005).
- [2] T. J. Sawik, "A scheduling algorithm for flexible flow lines with limited intermediate buffers", Applied Stochastic Models and Data Analysis, Vol. 9, No. 2, pp. 127-138 (1993).
- [3] L. X. Tang and H. Xuan, "Lagrangian relaxation algorithms for real-time hybrid flowshop scheduling with finite intermediate buffers", Journal of the Operational Research Society, Vol. 57, No. 3, pp. 316-324 (2006).
- [4] R. Tavakkoli-Moghaddam, N. Safaei, and F. Sassani, "A memetic algorithm for the flexible flow line scheduling problem with processor blocking", Computers and Operations Research, Vol. 36, No. 2, pp. 402-414 (2009).
- [5] X. Wang and L. Tang, "A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers", Computers and Operations Research, Vol. 36, No. 3, pp. 907-918 (2009).
- [6] B. Wardono and Y. Fathi, "A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities", European Journal of Operational Research, Vol. 155, No. 2, pp. 380-401 (2004).
- [7] R. J. Wittrock, "An adaptable scheduling algorithm for flexible flow lines", Operations Research, Vol. 36, No. 3, pp. 445-453 (1988).

Metaheuristic Approach to Loading Schedule Problem

Gregor Papa · Peter Korošec

Abstract The paper presents a comparison of two metaheuristic approaches in solving a constrained transportation scheduling problem (i.e., transport of goods in an emergency situations). We compared Parameter-less Evolutionary Search and Ant-Stigmergy Algorithm. The transportation scheduling/allocation problem is NP-hard, and is applicable to different real-life situations with high frequency of loading and unloading operations; like in depots, warehouses and ports.

1 Introduction

Transportation is a main component of supply chain and distribution between depots. Assigning and scheduling vehicle routing is a crucial management problem. Despite numerous publications on efficient scheduling methods for vehicle routing, very few address the loading schedule with multiple constraints; on different types of cargo, different vehicle capacities, different loading bay capacities, and overall transportation time minimization.

The vehicle routing problem (VRP) is a combinatorial optimization and nonlinear problem seeking to service a number of customers with a fleet of vehicles. VRP is an important problem in the fields of transportation, distribution and logistics [4, 6, 16, 26]. Usually, the goal is delivering goods located at a central depot to customers who have placed orders. This transportation optimization problem is NP-hard [15, 32], which means that the computational effort required to solve it increases exponentially with the problem size. To solve it in an acceptable time some stochastic algorithms are needed. Several papers cover vehicle routing by using different stochastic approaches, like simulated annealing [16], genetic algorithm [1, 24], ant-colony optimization [3, 22], and various hybrid methods [13, 31]

In this paper we describe a special version of VRP, where the vehicles circulate between two depots only. The stress is on optimal schedule of vehicles and allocations of vehicles to loading bays for loading and unloading of cargo, at the depots (see Figure 1), and can be treated as job shop scheduling problem. Here, each vehicle can

Computer Systems Department,
Jožef Stefan Institute, Jamova c. 39, SI-1000 Ljubljana, Slovenia
E-mail: {gregor.papa,peter.korosec}@ijs.si

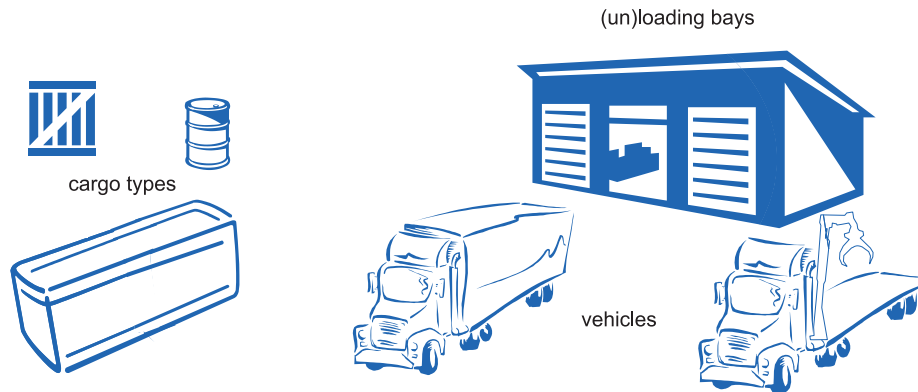


Fig. 1 Loading cargo at depot.

be represented as a resource and each visit to loading/unloading bay with a selected cargo type as an activity. Such transformation [2] proves that our problem is also NP-hard [10]. The case is especially useful in emergency/rescue situations (like floods, earthquakes, etc...), when large amounts of goods have to be transported from one depot to another as quickly as possible, with the limited number of available vehicles. Like VRP, this is also a combinatorial optimization problem. The emergency/rescue situations and disaster relief situations are covered in number of papers [9,14,19]; some of them also solve the problem with genetic algorithm [23,28] and ant-colony optimization [25,30].

To solve this loading schedule problem we used two metaheuristic techniques that have already proven as an efficient methods for solving combinatorial optimization problems. The first one was Parameter-Less Evolutionary Search (PLES) [20], and the second one was Ant-Stigmergy Algorithm (ASA) [12]. The PLES was already used in the process of the search for an optimal value of CEC'2006 functions, and in the process of geometry optimization of an electrical motor stator and rotor [20]. The ASA was used in the process of geometry optimization of an electrical motor stator and rotor, geometry optimization of radial impeller, and geometry optimization of electrical motor case [18,27].

The rest of the paper is organized as follows: in Section 2 we formally define the problem; in Section 3 we describe our approaches used for the search of the best schedule; in Section 4 we describe the experimentation results; and in Section 5 we list our conclusions.

2 Definitions

Our problem includes v vehicles, and c types of cargo (i.e., container, box, barrel, pallet...), with quantities V_j , where $j = 1, \dots, c$. For each vehicle we know the capacity for cargos w_{ij} , where $i = 1, \dots, v$ and $j = 1, \dots, c$. The capacity w_{ij} is measured in a cargo-relevant unit, i.e., ton, cubic meter, number of containers, etc. If the vehicle i is not capable of loading type j of a cargo, then its capacity is $w_{ij} = 0$.

Each vehicle can load in each run one type of a cargo only, however in different runs the same vehicle can load different cargos. The vehicle is always fully loaded; unless the remaining cargo is smaller than the capacity of the vehicle.

For each vehicle the maximum speed is known (for loaded and empty vehicle). With the distance between depot A and B we can calculate the driving times $t_{l1}, \dots, t_{lv}$ from A to B for loaded vehicles, and $t_{e1}, \dots, t_{ev}$ from B to A for empty vehicles.

At depot A there are m loading bays, each capable to load a_{zj} of cargo j per hour, $z = 1, \dots, m$. If the loading bay is not appropriate for type j of the cargo, then $a_{zj} = 0$. Similarly, at depot B there are n unloading bays, capable to unload b_{uj} of cargo j per hour, $u = 1, \dots, n$. There can be only one vehicle at a time on each loading or unloading bay.

A vehicle with self-loader is capable of loading a_{0j} of cargo j per hour, and unloading b_{0j} of cargo j per hour. The vehicle with a self-loader does not have to wait in a queue for the empty loading/unloading bay. It can be loaded/unloaded either at the appropriate and available loading bay, or outside the loading bay (e.g., somewhere at the warehouse/depot area).

The aim is to transport different types of cargo from depot A to B . Since the cargo can not be transported in one run, vehicles have to come back from B to A and perform several runs (cycles). Each run starts in A and consists of:

- waiting in queue till assigned loading bay is available,
- loading at loading bay,
- driving from A to B ,
- waiting in queue till assigned unloading bay is available,
- unloading at unloading bay,
- driving from B to A .

The scheduling operation consists of assigning (in each run) a type of a cargo for each vehicle, a loading bay and unloading bay. The aim is to find a schedule of cargo types and loading/unloading bays for all vehicles to finish the transportation of all cargos in shortest possible time. The general approach for solving this problem is represented in Algorithm 1.

Algorithm 1 A possible approach to solving the transportation problem

```

1: while all cargo not transported do
2:   for all vehicles do
3:     if current vehicle will be used then
4:       Choose among available cargo types for this vehicle.
5:       Reduce cargo according to vehicle's capacity.
6:       Choose among possible loading bays for a chosen cargo type.
7:       Choose among possible unloading bays for a chosen cargo type.
8:     end if
9:   end for
10: end while

```

The evaluation of each proposed schedule was done using a problem-specific simulator. The simulator followed the schedule and considered all the constraints, defined with the problem.

3 Algorithms

3.1 The PLES

The main advantage of Parameter-Less Evolutionary Search [20] over the basic GA [11] and other similar approaches is that the PLES can set the control parameters, like population size, number of generations, probabilities of crossover and mutation, by itself, during the optimization process. The values of parameters depend on the statistical behavior and convergence of the solutions. Elitism, selection, crossover and mutation are implemented through forcing of better individuals and moving of individuals. The control parameters are never set in advance and are not constant. The pseudo code of the algorithm is presented in Algorithm 2.

Algorithm 2 Parameter-Less Evolutionary Search

```

1: Set the initial population  $S$ .
2: Evaluate each individual of initial population.
3: Statistical calculations for initial population.
4: while stopping criterion (based on statistics) not met do
5:   Force better individuals to replace worse.
6:   Move each individual in the population.
7:   Evaluate each individual in the population.
8:   Statistical calculations for the population.
9: end while

```

3.1.1 Problem encoding

For v vehicles the chromosome looks like the string of $7v$ values, in general. Here, one value encodes the cargo type, and the next three pairs of values encode loading and unloading bay for each type of a cargo. If a vehicle is not capable to transport some type of a cargo, then the corresponding pair of loading/unloading bay values is omitted in the chromosome. So, for r runs with v vehicles we have a chromosome with $r \sum_{i=1}^v (1 + 2l_i)$ positions, where l_i is the number of possible cargos for vehicle i .

3.1.2 Optimization

The initial population size is proportional to chromosome size, i.e., problem complexity. The number of generations depends on the convergence of the global best solution. Optimization is stopped when the global best solution is not improved for some number of generations (see Eq. 1). The number depends on ratio of the initial population size ($InitPopSize$) and the number of generations since the last improvement ($Resting$) of the global best solution.

$$Limit = 10 \log_{10}(InitPopSize) + \log_{10}(Resting + 1) \quad (1)$$

In every generation the worse solutions are replaced with slightly permuted better solutions. Mutation is realized by moving of some positions in the chromosome according to statistical properties. The number of the positions in the chromosome to be moved depends on the standard deviation of the solution fitness of the previous

generation, and the maximal standard deviation of all generations. New value of the parameter is calculated upon the value of each parameter of the current solution, the value of parameter of the globally best solution, and the average value of the parameter in the previous generation.

3.1.3 Statistical calculations

Each population is statistically evaluated; the best, the worst, and average fitness value in the generation is found. Furthermore, the standard deviation of fitness values of all solutions in the generation, maximal standard deviation of fitness value over all generations, and average value of each parameter in the solution is calculated.

3.2 The ASA

The Ant-Stigmergy Algorithm [12] is an approach to solving multi-parameter optimization problems. It is based on stigmergy, a type of collective work that can be observed in ant colonies. The ASA consists of two main phases. The pseudo code of the algorithm is presented in Algorithm 3.

Algorithm 3 Ant-Stigmergy Algorithm

```

1: Construct the search graph from all parameters.
2: Initialize vertices with initial amount of pheromone.
3: while stopping criterion not met do
4:   for all ants do
5:     Find (using probability rule) the cheapest path.
6:   end for
7:   Update pheromone amounts in all vertices visited by the ants.
8:   Additionally increase the pheromone on currently best path.
9:   Evaporate pheromone in all vertices.
10: end while

```

3.2.1 Search graph construction

The problem parameters are transformed into a search graph where vertices represent all possible values of parameters. A vertex representing a parameter value is connected to all vertices representing the values of the next parameter. In our case, each vehicle run is represented with 3 parameters. First parameter represents types of cargo, second represents the possible loading bays and third represents the possible unloading bays. Due to this way of presentation, we have to search for subset of possible loading and unloading bays according to the possible types of cargo for a vehicle. For example, if vehicle could only carry cargos 1 and 3, than we need to select those loading and unloading bays that could manage both cargos. Of course this means that quite some number of possible solutions are omitted. Otherwise the algorithm is not capable of finding the feasible solution in reasonable time. So, for r runs with v vehicles we need a search graph with $3rv$ parameters. In this way, the constrained multi-parameter optimization problem is transformed into a problem of finding the cheapest path.

3.2.2 Optimization

Here the algorithm applies the optimization procedure based on ant-colony optimization [8]. All ants simultaneously start from the starting vertex. The probability of choosing the next vertex depends on the amount of pheromone in the vertices. Ants repeat this action until they reach the ending vertex. The parameter values gathered on each ant's path represent a candidate solution which is then evaluated according to the given objective function. Afterwards, each ant returns to the starting vertex, on its way depositing pheromone in the vertices according to the evaluation result: the better the result, the more pheromone is deposited. If the gathered parameter values form an infeasible solution, the amount of pheromone in the parameter vertices is slightly decreased. When the ants return to the starting vertex, two additional actions are performed. First, like in ant-colony optimization, a "daemon action" is applied as a kind of elitism, i.e., the pheromone amount on the currently best path is additionally increased. Second, the pheromone in all vertices evaporates, i.e., in each vertex the amount of pheromone is decreased by some predetermined percentage.

4 Performance evaluation

4.1 Experimental environment

The computer platform used to perform the experiments was based on Intel Pentium M770 2.13-GHz processor, 2 GB of RAM running Windows XP Professional operating system. The PLES was implemented in Microsoft Visual C++, while the ASA was implemented in Borland Delphi.

4.2 Experiments

Two algorithms (PLES and ASA) were used to find the shortest time for a task of transportation of cargos from depot *A* to *B*. The preliminary experiments, presented in this paper, consist of two tasks only. These tasks differ in properties of the loading bays. In the first task not all loading bays were capable of loading or unloading each cargo type, and in the second case all loading/unloading bays were capable of handling each cargo type. However, in both tasks for each loading/unloading bay the capacities vary for different cargo types.

For both experiments we had three types of a cargo, with quantities 30, 1,000 and 200,000, respectively, that were transported with 20 different vehicles. The vehicles had different speeds and different cargo capacities (see Table 1). Not all vehicles were able to transport every type of cargo and some vehicles also had self-loaders.

Table 1 Vehicle capacities for different cargo types

number of vehicles	2	4	6	4	4
cargo type 1	2	2	–	1	–
cargo type 2	38	50	12	8	8
cargo type 3	15,000	20,000	5,000	–	3,000
self-loader	no	no	yes	yes	no

Table 2 Loading/unloading bays capacities for different cargo types

	bay number	cargo type 1	cargo type 2	cargo type 3
loading	1	4	30	5,000
	2	-/5	50	5,000
	3	-/2	-/20	40,000
	4	4	35	4,000
	5	-/3	40	7,000
unloading	1	4	30	10,000
	2	-/5	50	5,000
	3	-/1	-/15	50,000
	4	3	35	4,000

At the depot A there were five loading bays, and at the depot B there were four unloading bays. The capacities of loading/unloading bays for different cargo types is presented in Table 2. While in task 2 there was no limitation on cargo types, its capacities are added in the Table 2, i.e. left value belongs to task 1 and the right one to task 2. Self-loading capacity was defined by the vehicles, and was always lower compared to the loading bay capacity.

We determined the lower bound of the number of runs for all vehicles. The lower bound was set according to the capacity of all cargos and capacities of all vehicles for each cargo. It is calculated as in Eq. 2.

$$r^* = \frac{V_1}{\sum_{i=1}^v w_{i1}} + \frac{V_2}{\sum_{i=1}^v w_{i2}} + \frac{V_3}{\sum_{i=1}^v w_{i3}} \quad (2)$$

This lower bound was used to determine the upper bound, which was used to narrow the search space and decrease the search time. The upper bound used for the PLES was $r = 1.5r^*$, while for the ASA the upper bound was set as $r = 2r^*$. The reason for using the upper bound was due to the fact that some vehicles can be omitted in some runs, therefore in general more runs are needed and only few solutions are feasible with $r = r^*$. With r set to some higher value the algorithm is able to quickly find more feasible solutions and then successfully converge to lower values, which are usually close to r^* .

Since the PLES is parameter-less algorithm, there was no need to set any control parameters. Stopping criteria was automatically set according to the convergence progress, i.e., approximately 30,000 evaluations for our experiment, or when maximal number of evaluations is reached, 500,000 evaluations. The ASA parameters were set as follows: the number of ants was 200, $\rho = 0.001$ and stopping criteria was determined by maximal number of evaluations, 500,000, or 50,000 evaluations without improvement. To set these parameters several test executions of the algorithm were needed.

Table 3 presents an extraction of the schedule for our experiment. The first column presents the vehicle to be used (it is not necessary that all vehicles are used in all runs), the second column presents the cargo type, while the third and the fourth column present the loading and unloading bay, respectively. Loading/unloading bay marked as 0 denotes the usage of the self-loader outside the loading bay.

Each algorithm was executed 20 times. Table 4 and Table 5, corresponding to task 1 and task 2 respectively, present the best, worst, mean, and standard deviation value of the transportation time (measured in hours). The fifth row presents standard error of the mean (SEM). The sixth row presents the confidence interval for the mean

Table 3 Example of the vehicle schedule and loading bay allocation

vehicle	cargo type	loading bay	unloading bay
1	3	3	3
2	1	4	4
3	1	4	1
5	2	2	2
7	2	1	0
9	2	1	4
10	3	4	2
13	1	0	1
17	3	2	3
19	2	1	2
20	3	5	4
2	3	3	3
5	2	5	2
6	1	4	4
7	2	0	4
9	2	0	4
12	2	5	0
...			
14	1	0	0
17	3	2	3
20	3	5	4

value with the confidence range of 99%. The last row presents the average number of evaluations needed to finish the algorithm run.

Table 4 Performance of the PLES and the ASA for task 1

	PLES	ASA
Best	19.70	24.60
Worst	23.36	27.97
Mean	21.07	26.00
St. Dev.	0.97	1.06
SEM	0.22	0.23
Conf. int. 99%	± 0.56	± 0.61
Avg. Evaluations	295,202	325,434

Table 5 Performance of the PLES and the ASA for task 2

	PLES	ASA
Best	20.82	18.38
Worst	24.76	21.73
Mean	22.17	19.90
St. Dev.	1.19	0.89
SEM	0.27	0.20
Conf. int. 99%	± 0.69	± 0.51
Avg. Evaluations	319,810	401,535

Table 6 presents a set of non-parametric statistical tests [5,7,17,21,29] for both algorithms in both tasks. The level of *F-test* shows that the results of both algorithms can be treated as having equal variance. According to *T-test* for two samples with equal

Table 6 Statistical test of the PLES and the ASA performance for task 1 and task 2

		task 1	task 2
<i>F-test</i>	p	0.71	0.61
<i>T-test</i>	t(38)	15.33	6.83
	p	<0.0001	<0.0001
<i>U-test</i>	z	-5.41	4.855
	U	0	20.5
	p	<0.0001	<0.0001

variance the difference between results of PLES and ASA algorithms is extremely statistically significant in both tasks with the level of significance $\alpha=0.01$. *T-test* requires normal distributions of samples, but since the size of the samples is relatively small (i.e., 20) the distribution shape could not be verified. Therefore, additional *U-test* was used to evaluate the results of a *T-test*. The results of *U-test* with $\alpha=0.01$ are again smaller than 0.0001, which additionally prove the statement above.

To obtain one solution the algorithms needed approximately 150,000 to 500,000 evaluations, which in time took 6 to 15 minutes. To transport the cargos, all vehicles had to perform about 7 runs.

Figure 2 and Figure 3 present the performance of both algorithms in task 1 and task 2, where the average, minimal and maximal values are presented in the graph.

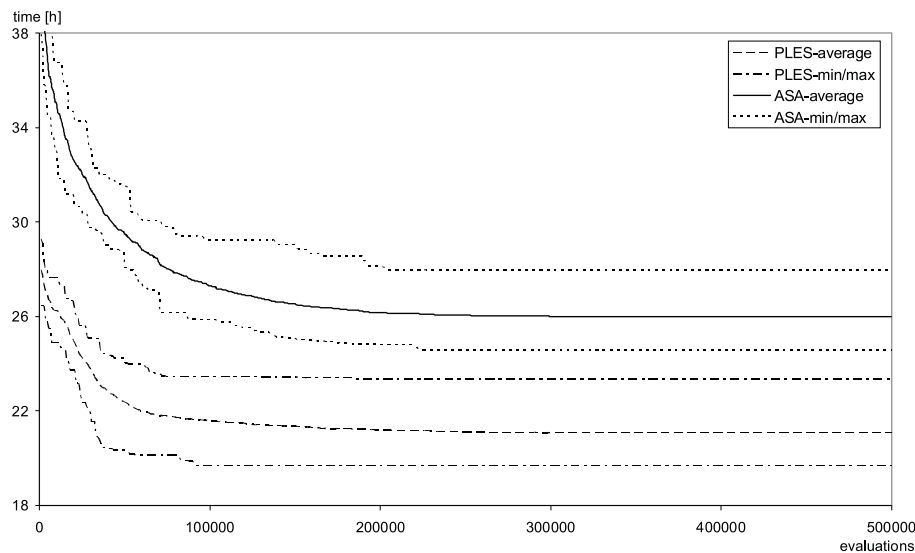


Fig. 2 Performance of the PLES and the ASA in task 1.

The reason for choosing two tasks that differ only in capacities of the loading and unloading bays is in the ASA. If we look at the results of the task 1, we notice that the PLES is far superior algorithm than the ASA. The reason for this is that with the PLES, we were able to fully utilize all bays capabilities, while with the ASA this was not possible. In general this means that with the PLES we could easily master all the

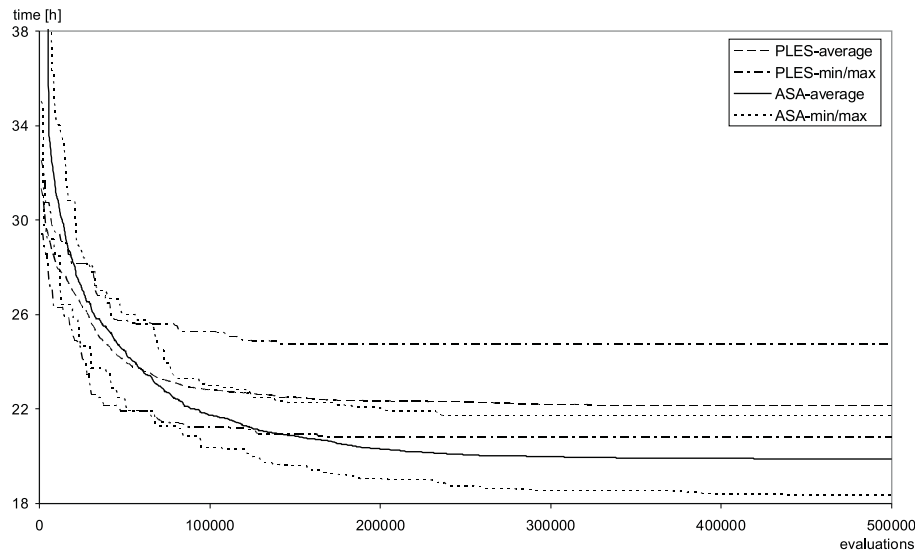


Fig. 3 Performance of the PLES and the ASA in task 2.

constrains that were opposed by the limiting loading/unloading bay capabilities. But with the ASA and its current graph construction process this was not possible and we had to make some simplifications to obtain feasible results. Therefore we found for each vehicle a subset of constraints that were true for all loading/unloading scenarios. For example, if one vehicle could load cargo 1 from bays 1 and 2 and cargo 2 from bays 2 and 3, we had to make a new constraint that is subset of both, which in this case means that this vehicle could load only cargos 1 and 2 and only from bay 2. If this was not done, the ASA algorithm was unable to find a good solution in allotted amount of time due to complexity of the problem. So just to test if the ASA would be capable of successfully solving this problem with a regular graph construction that would not call for such drastic measures we created task 2, where a vehicle can load/unload the cargos at all bays. The results of the task 2 show that the ASA is absolutely capable of successfully solving such problems and even producing better results than the PLES.

In task 2 the search space is increased (from approximately 10^{500} possible solutions in task 1 to 10^{583} possible solutions in task 2). It is obvious that the PLES was not capable to find better results due to the mentioned enormous increase. Probably it would be better to change the chromosome encoding in order to allow the algorithm to perform better exploration of the search space.

5 Conclusions

The results show that the PLES achieved better results than the ASA in task 1 due to constraint problems observed at the ASA graph construction process. In task 2 the ASA obtained better results then the PLES, where there was no such constraints. To get even better results with the PLES the encoding of the problem should be improved, i.e., chromosome length should be further decreased. This would improve the statistical

evaluation of solutions and improve the algorithm's convergence. As a result the search power of the algorithm would be increased. From the ASA results one can see that limiting the solution space is not the proper way to go. The constraints imposed with this prevented the ASA to find solutions that would be even close to PLES in task 1, while it excelled in task 2. In the future work we will try to remove this limit with the use of some kind of different search graph. In general, we will perform extended research with more complex task instances and deeper problem analyzing.

References

1. S.-T. Bae, H.S. Hwang, G.-S. Cho, and M.-J. Goan. Integrated GA-VRP solver for multi-depot system. *Computers & Industrial Engineering*, 53(2):233–240 (2007)
2. Beck, J.C., Prosser, P., Selensky, E.: Vehicle routing and job shop scheduling: Whats the difference. In *Proc. 13th International Conference on Artificial Intelligence Planning and Scheduling*, Trento, Italy, 267–276 (2003)
3. J.E. Bella and P.R. McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18 (1):41–48 (2004)
4. D.J. Bertsimas and G. Van Ryzin. A Stochastic and Dynamic Vehicle Routing Problem in the Euclidean Plane. *Operations Research* 39(4):601–615 (1991)
5. S. Brandt. *Statistical and Computational Methods in Data Analysis*. North Holland (1983)
6. G.B. Dantzig and J.H. Ramser. The Truck Dispatching Problem. *Management Science*, 6(1):80–91 (1959)
7. J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30 (2006)
8. M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172 (1999)
9. K.F. Doerner and R.F. Hartl. Health Care Logistics, Emergency Preparedness, and Disaster Relief: New Challenges for Routing Problems with a Focus on the Austrian situation. In B. Golden et al. (eds): *The Vehicle Routing Problem : Latest Advances and New Challenges*, pp. 527–550. (2008)
10. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company (1979)
11. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley (1989)
12. P. Korošec and J. Šilc. The multilevel ant stigmergy algorithm: an industrial case study. In *Proc. 7th Int. Conf. Computational Intelligence and Natural Computing*, Salt Lake City, UT, pp: 475–478 (2005)
13. C.-Y. Lee, Z.-J. Lee, S.-W. Lin, and K.-C. Ying. An enhanced ant colony optimization (EACO) applied to capacitated vehicle routing problem. *Applied Intelligence*, DOI 10.1007/s10489-008-0136-9 (2008)
14. F. Lei. Dynamic multi-objective emergency relief logistics: A decision support system framework. In *Proc. Grey Systems and Intelligent Services, GSIS 2007*, Nanjing, China, pp: 779–783 (2007)
15. J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks* 11 (2):221–227 (1981)
16. S.-W. Lin, V.F. Yu, and S.-Y. Chou. Solving the truck and trailer routing problem based on a simulated annealing heuristic. *Computers & Operations Research*, 36 (5): 1683–1692 (2009)
17. H.B. Mann, D.R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics*, 18:50–60 (1947)
18. K. Oblak, P. Korošec, F. Kosel, and J. Šilc. Multi-parameter numerical optimization of selected thin-walled machine elements using a stigmergic optimization algorithm. *Thin-Walled Structures*, 45(12):991–1001 (2007)
19. L. Özdamar and W. Yi. Greedy Neighborhood Search for Disaster Relief and Evacuation Logistics. *IEEE Intelligent Systems*, 23 (1):14–23 (2008)
20. G. Papa. Parameter-Less Evolutionary Search. In *Proc. Genetic and Evolutionary Computation Conference (GECCO'08)*, Atlanta, GE pp: 1133–1134 (2008)

21. W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press (1992)
22. A.E. Rizzoli, R. Montemanni, E. Lucibello, and L.M. Gambardella. Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence*, 1(2): 135-151 (2007)
23. M. Saadatseresht, A. Mansourian, and M. Taleai. Evacuation planning using multiobjective evolutionary optimization approach. *European Journal of Operational Research*, 198(1):305–314 (2009)
24. S. Salhi and R. Petch. A GA based heuristic for the vehicle routing problem with multiple trips. *Journal of Mathematical Modelling and Algorithms*, 6(4):591–613 (2007)
25. B. Tatomir and L. Rothkrantz. Ant Based Mechanism for Crisis Response Coordination. In M. Dorigo et al. (Eds.): *ANTS 2006, LNCS 4150*, pp. 380-387 (2006)
26. P. Toth. *The Vehicle Routing Problem*. Philadelphia: Siam. (2001)
27. T. Tušar, P. Korošec, G. Papa, B. Filipič, and J. Šilc. A comparative study of stochastic optimization methods in electric motor design. *Applied Intelligence*, 27(2):101–111 (2007)
28. X. Wang and H. Wang. Designing Optimal Emergency Logistics Networks with Time Delay and Demand Uncertainty. In *Proc 19th Annual Conference of the Production and Operations Management Society, POM 2008*, La Jolla, California, (2008)
29. F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80-83 (1945)
30. W. Yi and A. Kumar. Ant colony optimization for disaster relief operations. *Transportation Research Part E: Logistics and Transportation Review*, 43 (6):660–672 (2007)
31. E.E. Zachariadis, C.D. Tarantilis, and C.T. Kiranoudis. A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service. *Expert Systems with Applications*, 36 (2):1070–1081 (2009)
32. L. Zeng, H.L. Ong, and K.M. Ng. A generalized crossing local search method for solving vehicle routing problems. *Journal of the Operational Research Society*, 58:528-532. (2007)

Generating Cyclic Fair Sequences for Multiple Servers

Jeffrey W. Herrmann

Abstract Fair sequences allocate capacity to competing demands in a variety of manufacturing and computer systems. This paper considers the generation of cyclic fair sequences for a set of servers that must process a given set of products, each of which must be produced multiple times in each cycle. The objective is to create a sequence that minimizes, for each product, the variability of the time between consecutive completions (the response time). Because minimizing response time variability is known to be NP-hard and the performance of existing heuristics is poor for certain classes of problems, we present an aggregation approach that combines products with the same demand into groups, creates a sequence for those groups, and then disaggregates the sequence into a sequence for each product. Computational experiments show that using aggregation can reduce computational effort and response time variability dramatically.

1 Introduction

In many applications, it is important to schedule the resource's activities in some fair manner so that each demand receives a share of the resource that is proportional to its demand relative to the competing demands. This problem arises in a variety of domains. A mixed-model, just-in-time assembly line produces different products at different rates. Maintenance and housekeeping staff need to perform preventive maintenance and housekeeping activities regularly, but some machines and areas need more attention than others. Vendors who manage their customers' inventory must deliver material to their customers at rates that match the different demands. Asynchronous transfer mode (ATM) networks must transmit audio and video files at rates that avoid poor performance.

The idea of fair sequences occurs in many different areas, including those mentioned above. Kubiak [1] reviewed the need for fair sequences in different domains and concluded that there is no single definition of a "fair sequence." This review discussed results for multiple related problems, including the product rate variation problem, generalized pinwheel scheduling, the hard real-time periodic scheduling problem, the periodic maintenance scheduling problem, stride scheduling, minimizing response time variability (RTV), and peer-to-peer fair scheduling. Bar-Noy et al. [2] discussed the generalized maintenance scheduling problem, in which the objective is to minimize the long-run average service and operating cost per time slot. Bar-Noy et al. [3] considered the problem of creating a schedule in which a single broadcast channel transmits each page perfectly periodically (that is, the interval

Jeffrey W. Herrmann
University of Maryland, College Park
E-mail: jwh2@umd.edu

between consecutive transmissions of a page is always the same). The goal is to find a feasible schedule so that the lengths of the scheduled periods are close to the ideal periods that will minimize average waiting time.

In the periodic maintenance scheduling problem, the intervals between consecutive completions of the same task must equal a predetermined quantity. Wei and Liu [4] presented sufficient conditions for the existence of a feasible solution for a given number of servers. They also suggested that machines with the same maintenance interval could be replaced by a substitute machine with a smaller maintenance interval and that this replacement would facilitate finding a feasible solution. This concept, which was not developed into a solution algorithm, is similar to the aggregation proposed here. Park and Yun [5] and Campbell and Hardin [6] studied other problems that require the activities to be done strictly periodically and developed approaches to minimize the number of required resources. Park and Yun considered activities such as preventive maintenance tasks that take one time unit but can have different periods and different resource requirements. Campbell and Hardin studied the delivery of products to customers, where each delivery takes one day but the customers have different replenishment periods that must be satisfied exactly.

Waldspurger and Weihl [7] considered the problem of scheduling multithreaded computer systems. In such a system, there are multiple clients, and each client has a number of tickets. A client with twice as many tickets as another client should be allocated twice as many quanta (time slices) in any given time interval. Waldspurger and Weihl introduced the stride scheduling algorithm to solve this problem. They also presented a hierarchical stride scheduling approach that uses a balanced binary tree to group clients, uses stride scheduling to allocate quanta to the groups, and then, within each group, using stride scheduling to allocate quanta to the clients. Although they note that grouping clients with the same number of tickets would be desirable, their approach does not exploit this. Indeed, the approach does not specify how to create the binary tree.

Kubiak [1] showed that the stride scheduling algorithm is the same as Jefferson's method of apportionment and is an instance of the more general parametric method of apportionment [8]. Thus, the stride scheduling algorithm can be parameterized.

The problem of mixed-model assembly lines has the same setup as the RTV problem but a different objective function. Miltenburg [9] considered objective functions that measure (in different ways) the total deviation of the actual cumulative production from the desired cumulative production and presented an exact algorithm and two heuristics for finding feasible solutions. Inman and Bulfin [10] considered the same problem and assigned due dates to each and every unit of the products. They then presented the results of a small computational study that compared earliest due date (EDD) schedules to those generated using Miltenburg's algorithms on a set of 100 instances. They show that the EDD schedules have nearly the same total deviation but can be generated much more quickly.

A more flexible approach in a cyclic situation is to minimize the variability in the time between consecutive allocations of the resource to the same demand (the same product or client, for instance). Corominas et al. [11] presented and analyzed the RTV objective function (which will be defined precisely below), showed that the RTV problem is NP-hard, and presented a dynamic program and a mathematical program for finding optimal solutions. Because those approaches required excessive computational effort, they conducted experiments to evaluate the performance of various heuristics. However, the Webster and Jefferson heuristics (described below) reportedly performed poorly across a range of problem instances. Garcia et al. [12] presented metaheuristic procedures for the problem. Kubiak [13] reviews these problems and others related to apportionment theory.

Independently, Herrmann [14] described the RTV problem and presented a heuristic that combined aggregation and parameterized stride scheduling. The aggregation approach combines products with the same demand into groups, creates a sequence for those groups, and then disaggregates the sequence into a sequence for each product.

For the single-server RTV problem, Herrmann [15] precisely defined the aggregation approach and described the results of extensive computational experiments using the

aggregation approach in combination with the heuristics presented by [11]. The results showed that solutions generated using the aggregation approach can have dramatically lower RTV than solutions generated without using the aggregation approach. Herrmann [16] considered a more sophisticated “perfect aggregation” approach that can generate zero RTV sequences for some instances.

In this work we consider the problem when multiple servers, working in parallel, are available. Although the problem has some similarities to the problem of scheduling real-time tasks on multiprocessors, in the problem studied here, we will generate a complete schedule for the entire cycle (instead of relying on priority rules to determine which task to perform at each point in time), and our objective is to reduce response time variability (not to meet time windows on each subtask). The purpose of this paper is to determine how the aggregation technique performs in combination with heuristics for the multiple-server RTV problem. The current paper thus extends previous work on the RTV problem (which considered only a single server).

The remainder of this paper describes the problem formulation, presents the aggregation approach, discusses the heuristics and a lower bound, and presents the results of extensive computational experiments using the aggregation approach in combination with heuristics developed for this problem.

2 Problem Formulation

Given M servers that must produce n products, each with a demand d_i that is a positive integer, let $D = d_1 + \dots + d_n$. (We assume that M divides D , as we can add sufficient dummy products with $d_i = 1$ to pad the instance.) Each product requires the same amount of time, so we can ignore time and consider only the positions in the sequence. A feasible sequence $s = s_1 s_2 \dots s_L$ has length $L = D/M$ and specifies the M products that are served in each of the L positions. That is, for $j = 1, \dots, L$, each s_j is a subset of $\{1, \dots, n\}$ with exactly M elements. Each product i occurs exactly d_i times in the sequence. No product can be processed by two servers at the same time. (Thus, all $d_i \leq L$.) Moreover, this sequence will be repeated, and we will call each occurrence a cycle. The RTV of a feasible sequence equals the sum of the RTV for each product. If product i occurs at positions $\{p_{i1}, \dots, p_{id_i}\}$, its RTV is a function of the intervals between each position, which are $\{\Delta_{i1}, \dots, \Delta_{id_i}\}$, where the intervals are measured as $\Delta_{ik} = p_{ik} - p_{i,k-1}$ (with $p_{i0} = p_{id_i} - L$). The average interval for product i is L/d_i , so we calculate RTV as follows:

$$RTV = \sum_{i=1}^n \sum_{k=1}^{d_i} \left(\Delta_{ik} - \frac{L}{d_i} \right)^2$$

Thus, we can describe the problem as follows: Given an instance $\{d_1, \dots, d_n\}$, find a sequence s of length L that minimizes RTV subject to the constraints that exactly M different products are served in each and every one of the L positions and each and every product i occurs exactly d_i times in s .

The single-server version of this problem is NP-hard [11], and it is easy to see that the multiple-server version must be NP-hard as well.

Consider, as an example, the following nine-product instance: $\{1, 1, 2, 2, 2, 2, 2, 3, 5\}$. Table 1 lists a feasible sequence for this instance; its RTV equals 16.67.

Table 1. A feasible sequence for a two-server, nine-product RTV problem.
(The product demands are {1, 1, 2, 2, 2, 2, 2, 3, 5}.)

Server										
1	9	3	5	7	1	8	9	4	6	8
2	8	4	6	9	2	9	3	5	7	9

3 Aggregation

Aggregation is a well-known and valuable technique for solving optimization problems, especially large-scale mathematical programming problems. Model aggregation replaces a large optimization problem with a smaller, auxiliary problem that is easier to solve [17]. The solution to the auxiliary model is then disaggregated to form a solution to the original problem. Model aggregation has been applied to a variety of production and distribution problems, including machine scheduling problems. In the single-server RTV problem, iteratively combining products with the same demand transforms the original instance into one with fewer products. This aggregation can lead to dramatically better solutions [15]. This was first introduced in [14] and is similar to the substitution concept discussed by [4]. This paper presents a version of this aggregation that is appropriate for multiple servers.

For a set of products that all have the same demand, the positions in a sequence where those products occur should be assigned to those products in a round-robin fashion. Thus, for that set of products, the challenge of assigning positions to those products is equivalent to assigning positions to the set as a whole. Once an appropriate number of positions has been assigned to the entire set, it is easy to determine which products should go in which positions.

The aggregation combines products that have the same demand into groups. This has two benefits. First, it reduces the number of products that need to be sequenced. Secondly, because a group will have higher demand, it will be blended better than the individual products. Nevertheless, the demand of a group must be no greater than the cycle length L .

For instance, consider the nine-product example presented above. Combining the products 3 to 7, which all have a demand of 2, into one group with a demand of 10, simplifies the problem significantly. Because the sequence length equals 10, it is clear that this group should be served in every position. From there, it is a straightforward task to assign positions 1 and 6 to one of the five products, positions 2 and 7 to a second product, and so forth.

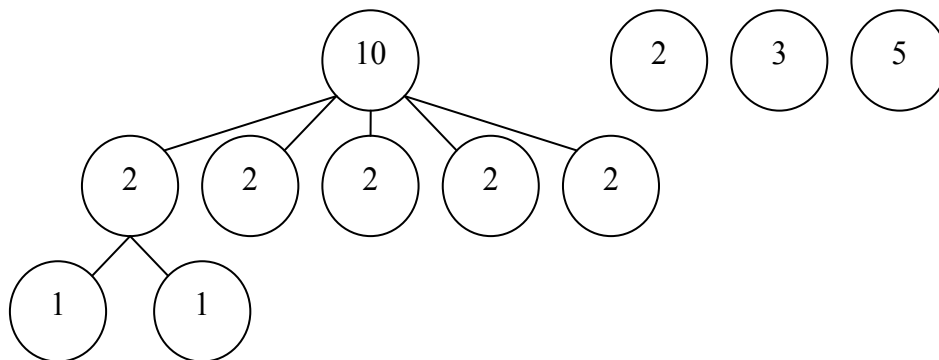


Figure 1. The forest of weighted trees corresponding to the aggregation of the two-server, nine-product example. The number in each node is the demand of the corresponding product.

The aggregation approach presented here repeatedly aggregates an instance until it cannot be aggregated any more. In our example, the first aggregation combines the first two products into a group with a demand of 2. This generates a new instance that has six products with a demand of 2. We cannot combine all six products because the sequence length equals only 10.

Thus, the second aggregation combines five to these products to form a group with a demand of 10. This generates a second new instance that has only four products, and their demands are $\{2, 3, 5, 10\}$. Because they all have different demands, no more aggregation is possible. As discussed in more detail below, an aggregation can be visualized as a forest, as shown in Figure 1.

The aggregation procedure generates a sequence of instances $I_0, \dots, I_H$. (H is the index of the last aggregation created.) The aggregation can be done at most $n_0 - 1$ times because the number of products decreases by at least one each time the aggregation procedure is called (unless no aggregation occurs). Thus $H \leq n_0 - 1$.

The more complex notation used in the following steps enables us to keep track of the aggregations in order to describe the disaggregation of a sequence precisely. Let I_0 be the original instance and I_k be the k -th instance generated from I_0 . Let n_k be the number of products in instance I_k . Let P_j be the set of products that form the new product j , and let $P_j(i)$ be the i -th product in that set.

Aggregation. Given: an instance I_0 with demands $\{d_1, \dots, d_n\}$ and M servers.

1. Initialization. Let $k = 0$ and $n_0 = n$.
2. Stopping rule. If all of the products in I_k have different demands, return I_k and $H = k$ because no further aggregation is possible. Otherwise, let G be the set of products with the same demand such that any smaller demand is unique.
3. Aggregation.
 - 3a. Let i be one of the products in G . If $|G| > L/d_i$, then G is too large. Keep $\lfloor L/d_i \rfloor$ products and remove any others.
 - 3b. Let $m = |G|$. Create a new product $n + k + 1$ with demand $d_{n+k+1} = md_i$. Create the new instance I_{k+1} by removing from I_k all m products in G and adding product $n + k + 1$. Set $P_{n+k+1} = G$ and $n_{k+1} = n_k - m + 1$. Increase k by 1 and go to Step 2.

Consider the nine-product example presented above. The first aggregation creates instance I_1 by combining two products of I_0 ($G = \{1, 2\}$) and creating product 10 with $P_{10} = \{1, 2\}$ and $d_{10} = 2$. The new instance has $n_1 = 8$ products.

The second aggregation creates instance I_2 by combining the five products of I_1 ($G = \{3, 4, 5, 6, 10\}$) and creating product 11 with $P_{11} = \{3, 4, 5, 6, 10\}$ and $d_{11} = 10$. (Note that product 7 cannot be included due to the length constraint.) The new instance has $n_2 = 4$ products. No more aggregation is possible, so $H = 2$. See also Table 2 and Figure 1.

Table 2. The product sets for the original and new instances.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}
I_0	1	1	2	2	2	2	2	3	5		
I_1			2	2	2	2	2	3	5	2	
I_2							2	3	5		10

The total demand in each new instance will equal the total demand of the original instance because the demand of the new product (which we call a “group”) equals the total demand of the products that were removed.

We can represent the aggregation as a forest of weighted trees (as shown in Figure 1). There is one tree for each product in I_H . The weight of the root of each tree is the total demand of the products that were aggregated to form the corresponding product in I_H . The

weight of any node besides the root node is the weight of its parent divided by the number of children of the parent. The leaves of a tree correspond to the products in I_0 that were aggregated to form the corresponding product in I_0 , and each one's weight equals the demand of that product. The forest has one parent node for each group formed during the aggregation, and the total number of nodes in the forest equals $n_0 + H < 2n_0$.

Let S_H be a feasible sequence for instance I_H . Disaggregating S_H requires H steps that correspond to the aggregations that created the new instances. We disaggregate S_H to generate S_{H-1} and then continue to disaggregate each sequence in turn to generate $S_{H-2}, \dots, S_0$. S_0 is a feasible sequence for I_0 , the original instance.

The basic idea of disaggregating a sequence is to replace each product from that instance with the corresponding product (or products) from the less-aggregate instance. Consider, for instance, the example presented above and S_2 , a feasible sequence for I_2 , shown as the first sequence in Table 3. Because product 11 in I_2 was formed by combining five products in I_1 (the products in P_{11}), the 10 positions in S_2 that are assigned to that product will be given to those five products. This disaggregation yields the sequence S_1 , a feasible sequence for I_1 , shown as the second sequence in Table 3.

This sequence, in turn, can be disaggregated to form S_0 , a feasible sequence for I_0 , shown as the third sequence in Table 3. Because product 10 in I_1 was formed by combining products 1 and 2 in I_0 , the positions in S_1 that are assigned to that product will be given to those two products.

Table 3. Initial and disaggregated solutions.

Server										
1	11	11	11	11	11	11	11	11	11	11
2	9	7	9	8	9	7	9	8	9	8
1	3	4	5	6	10	3	4	5	6	10
2	9	7	9	8	9	7	9	8	9	8
1	3	4	5	6	1	3	4	5	6	2
2	9	7	9	8	9	7	9	8	9	8

In the following $j = S_k(s, a)$ means that server s processes product j in position a of sequence S_k . $P_{n+k}(i)$ is the i -th product in P_{n+k} .

Disaggregation. Given: The instances $I_0, \dots, I_H$ and the sequence S_H , a feasible sequence for instance I_H .

1. Initialization. Let $k = H$.
2. Set $m = |P_{n+k}|$ and $i = 1$.
3. For $a = 1, \dots, L$, perform the following step:
 - a. For $s = 1, \dots, M$, perform the following step:
 - i. Let $j = S_k(s, a)$. If $j < n+k$, then assign $S_{k-1}(s, a) = j$. Otherwise, assign $S_{k-1}(s, a) = P_{n+k}(i)$, increase i by 1, and, if $i > m$, set $i = 1$.
4. Decrease k by 1. If $k > 0$, go to Step 2. Otherwise, stop and return S_0 .

Aggregation runs in $O(n^2)$ time because each aggregation requires $O(n)$ time and there are at most $n-1$ aggregations. Likewise, because each sequence disaggregation requires $O(D)$ effort, disaggregation runs in $O(nD)$ time in total.

The key distinctions between the hierarchical stride scheduling algorithm of [7] and the aggregation approach presented here are (1) the hierarchical stride scheduling algorithm requires using the stride scheduling algorithm to disaggregate each group, since the products in a group may have unequal demands and (2) the placement of products in the tree is not specified. Because our aggregation scheme groups products with equal demand, the disaggregation is much simpler. The limitation, however, is that the instance must have some equal demand products.

Also, unlike [3], our aggregation approach does not adjust the demands so that they fit into a single weighted tree (which would lead to a perfectly periodic schedule). Instead, the demands are fixed, and the aggregation creates a forest of weighted trees, which are used to disaggregate a sequence.

4 Heuristics

The purpose of this paper is to investigate how much the aggregation approach reduces RTV when used with various heuristics for the multi-server problem. This section presents the heuristics that will be used. There are four basic heuristics (which construct sequences) and an improvement heuristic that manipulates sequences.

The parameterized stride scheduling algorithm builds a fair sequence and performs well at minimizing the maximum absolute deviation [1]. The algorithm has a single parameter δ that can range from 0 to 1. This parameter affects the relative priority of low-demand products and their absolute position within the sequence. When δ is near 0, low-demand products will be positioned earlier in the sequence. When δ is near 1, low-demand products will be positioned later in the sequence. Here we describe a multiple-server version of the algorithm.

Parameterized stride scheduling. Given: an instance $\{d_1, \dots, d_n\}$, M servers, and the parameter δ .

1. Initialization. $x_{i0} = 0$ for $i = 1, \dots, n$. (x_{ik} is the number of times that product i occurs in the first k positions of the sequence.)
2. For $k = 0, \dots, D-1$, perform the following steps:
 - a. For all products i that satisfy $d_i - x_{ik} = L - k$, assign those products to servers in position $k + 1$ (because the remaining demand for the product equals the number of remaining positions in the sequence).
 - b. Any unassigned servers are assigned the products with the greatest values of the ratio $d_i / (x_{ik} + \delta)$. In case of a tie, select the lowest-numbered product to break a tie.
 - c. For all products i that were assigned to a server, set $x_{i,k+1} = x_{i,k} + 1$. For all other products, set $x_{i,k+1} = x_{i,k}$.

The computational effort of the algorithm is $O(nD)$. Table 1 shows the solution generated by the parameterized stride scheduling algorithm (with $\delta = 0.5$) for the nine-product instance introduced earlier. The RTV equals 16.67. (Other values of δ change the relative positions of some products but do not affect the RTV.) Note that products 3 to 7 (which all have the same demand) are grouped together first in positions 2 to 4 and again in positions 7 to 9, which leads to high RTV for the higher-demand products.

Two versions of the parameterized stride scheduling algorithm are well-known apportionment methods. Following [11], we will use the names “Webster” and “Jefferson” to denote these heuristics. These names refer to the traditional parametric methods of apportionment to which they are equivalent [1, 8]. The Webster heuristic uses $\delta = 0.5$. The Jefferson heuristic uses $\delta = 1$. Clearly, both the Webster and Jefferson heuristics run in $O(nD)$ time.

The bottleneck heuristic is the due date algorithm that Steiner and Yeomans [18] developed for solving the mixed-model production problem, modified to schedule multiple servers. This heuristic runs in $O(D \log D)$ time. Here we will describe the algorithm in detail.

Bottleneck. Given: an instance $\{d_1, \dots, d_n\}$ and M servers.

1. Set $w = 0$ and $x_i = 1$ for all $i = 1, \dots, n$.
2. For $i = 1, \dots, n$ and $j = 1, \dots, d_i$, calculate the following quantities:

$$EST_{ij} = \left\lceil \frac{L(j-1) + w}{d_i} - 1 \right\rceil$$

$$LST_{ij} = \left\lfloor \frac{Lj - w}{d_i} \right\rfloor$$

3. For $k = 0, \dots, L-1$, perform the following steps:
 - a. Let $R = \{i : x_i \leq d_i, EST_{ix_i} \leq k, LST_{ix_i} \geq k\}$.
 - b. If $|R| \geq M$, then let R' be the M products in R that have the smallest LST_{ix_i} . Otherwise, go to Step 5.
 - c. For all i in R' , assign one task of product i to position $k+1$ on one server, and add one to x_i .
4. Save the current sequence. If $w < \max\{d_i\}$, then set w to the value of the smallest d_i that is greater than w , and go to Step 2. Otherwise, go to Step 5.
5. Return the last saved sequence.

The sequential heuristic is an extremely simple sequence construction approach. It builds the sequence one product at a time, assigning all of the positions for one server and then moving to the next one. (It is similar to scheduling preemptive tasks on parallel machines.)

Sequential. Given: an instance $\{d_1, \dots, d_n\}$ and M servers.

1. Set $s = 1$ and $a = 1$.
2. For $i = 1, \dots, n$, perform the following step:
 - a. For $j = 1, \dots, d_i$ perform the following steps:
 - i. Assign product i to server s in position a .
 - ii. Increase a by 1. If $a > L$, set $s = s + 1$ and $a = 1$.

This heuristic runs in $O(D)$ time. Table 5 shows the solution generated by the sequential for the nine-product instance introduced earlier. The RTV equals 212.67.

Table 5. Solution generated by the sequential heuristic.

Server										
1	1	2	3	3	4	4	5	5	6	6
2	7	7	8	8	8	9	9	9	9	9

The exchange heuristic is a technique for improving any given sequence [11]. It repeatedly loops through the positions, exchanging a task with the task immediately following it if that exchange reduces the RTV or reduces the maximum distance between two tasks for a product. It runs in $O(nD^4)$ time. We modified the exchange heuristic for the multiple server case by restricting exchanges to those that strictly reduce the RTV and finding, among all possible exchanges between two consecutive positions, the one that most reduces RTV.

Exchange. Given: an instance $\{d_1, \dots, d_n\}$, M servers, and a sequence S .

1. Set $flag = 0$.
2. For $a = 1, \dots, L$, perform the following steps:
 - a. If $a < L$, set $b = a+1$. Otherwise, set $b = 1$.
 - b. For $s = 1, \dots, M$, perform the following step:

- i. Set $i = S(s, a)$. Set L_i equal to the distance between position a and the last position in which product i occurs before position a (going back to the last cycle if needed). Set R_i equal to the distance between position a and the next position in which product i occurs after position a (going on to the next cycle if needed). If $d_i = 1$, set $E_i = 0$; otherwise, set $E_i = 2(R_i - L_i - 1)$, which is the decrease in product i 's RTV if product i is moved to position b . If product i already occurs in position b , then this move is not feasible.
- c. Find the product i^* among those that occur in position a that gives the largest value of E_i and can be moved feasibly to position b .
- d. For $s = 1, \dots, M$, perform the following step:
 - i. Set $j = S(s, b)$. Set L_j equal to the distance between position b and the last position in which product j occurs before position b (going back to the last cycle if needed). Set R_j equal to the distance between position b and the next position in which product j occurs after position b (going on to the next cycle if needed). If $d_j = 1$, set $F_j = 0$; otherwise, set $F_j = 2(L_j - R_j - 1)$, which is the decrease in product j 's RTV if product j is moved to position a . If product j already occurs in position a , then this move is not feasible.
 - e. Find the product j^* among those that occur in position b that gives the largest value of F_j and can be moved feasibly to position a .
 - f. If $E_{i^*} + F_{j^*} > 0$, then exchange i^* and j^* by moving i^* to position b and moving j^* to position a , and set $flag = 1$.
3. If $flag = 1$, then go to Step 1. Otherwise, stop and return the current sequence.

For instance, consider our nine-product example and the sequence shown in Table 5. Suppose that the exchange heuristic is considering position 1. When $a = 1$, product 7 already occurs in position 2. Thus, $i^* = 1$. Likewise, $j^* = 2$. $E_1 + F_2 = 0 + 0 = 0$, so no exchange occurs. So the heuristic moves to position $a = 2$. Here, $E_2 = 0$ and $E_7 = 2(9-1-1) = 14$, so $i^* = 7$. With $b = 3$, $F_3 = 2(9-1-1) = 14$ and $F_8 = 2(8-1-1) = 12$, so $j^* = 3$. Because $E_{i^*} + F_{j^*} > 0$, the heuristic exchanges products 7 and 3 and then goes on to $a = 3$.

5 Lower Bound

To provide some idea of the quality of the solutions generated, we will compare their RTV to a lower bound. The lower bound on the total RTV is the sum of lower bounds for each product's RTV. This lower bound depends on product i 's demand and the total sequence length L . Let LB_i be the lower bound on the RTV of product i , which can be calculated as follows by considering intervals that are just shorter and longer than the average:

1. Set $a_i = L \bmod d_i$ (the number of intervals that will be longer than the average) and $A_i = \lfloor L / d_i \rfloor$ (the largest integer less than or equal to L / d_i , which equals the length of the shorter interval) and $B_i = A_i + 1$ (the length of the longer interval).
2. Set $LB_i = (d_i - a_i)(A_i - L / d_i)^2 + a_i(B_i - L / d_i)^2$.

We note that, if d_i divides L , $LB_i = 0$. The lower bound LB on the total RTV is the sum of the product lower bounds: $LB = LB_1 + \dots + LB_n$.

6 Computational Experiments

The purpose of the computational experiments was to determine how the aggregation technique performs in combination with a variety of heuristics on the metric of RTV. All of the algorithms were implemented in Matlab and executed using Matlab R2006b on a Dell Optiplex GX745 with Intel Core2Duo CPU 6600 @ 2.40 GHz and 2.00 GB RAM running Microsoft Windows XP Professional Version 2002 Service Pack 3.

We considered two values for the number of servers: $M = 5$ and $M = 10$. We used 3900 of the 4700 instances described in [15]. These were generated as follows: First, we set the total number of tasks D and the number of products n . (We followed [11] in choosing values of D and n , but these are not the same instances.) To generate an instance, we generated $D - n$ random numbers from a discrete uniform distribution over $\{1, \dots, n\}$. We then let d_i equal 1 plus the number of copies of i in the set of $D - n$ random numbers (this ensured that each d_i is at least 1). We generated 100 instances for each problem set (a combination of D and n). In this study, we used the following problem sets: with $D = 100$, $n = 40, 50, 60, 70, 80$, and 90 ; with $D = 500$, $n = 50, 100, 150, 200, 250, 300, 350, 400$, and 450 ; with $D = 1000$, $n = 50, 100, 200, 300, 400, 500, 600, 700, 800$, and 900 ; and with $D = 1500$, $n = 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300$, and 1400 .

For each instance and value of M , we constructed 16 sequences as follows. First, we applied one of the basic heuristics to the instance (we call this the H sequence). Then, we applied the exchange heuristic to the H sequence to construct the HE sequence.

Next, we aggregated the instance. For the aggregate instance, we applied the heuristic to construct an aggregated solution. We disaggregated this solution to construct the AHD sequence. Then, we applied the exchange heuristic to the AHD sequence to construct the AHDE sequence. This makes four sequences using one basic heuristic and combinations of aggregation-disaggregation and the exchange heuristic. We repeated this for the remaining basic heuristics for a total of 16 solutions.

Before discussing the results of the heuristics, we note that all of the instances considered could be aggregated at least once. For a given value of D , the average number of times that an instance was aggregated increased as D increased but decreased as the number of products n increased. For $M = 5$, $D = 100$, and $n = 90$, the average number of aggregations was 5.1. For $M = 5$, $D = 1500$, and $n = 100$, the average number of aggregations was 14.8.

For $M = 10$, the average number of aggregations exhibited the same trends, and ranged from 10.1 (with $D = 100$ and $n = 90$) to 17.3 (with $D = 1500$ and $n = 100$). More aggregations were possible due to the shorter sequence length.

Compared to the single-server problem [15], the number of aggregations is generally larger because the cycle length restriction limits the number of products that can be combined into a group. That is, instead of combining many products with the same demand into one large group, the aggregation must form multiple groups that are not too long, and this increases the number of aggregations.

Because the exchange heuristic is known to reduce RTV, our discussion will focus on the HE and AHDE sequences. Tables 6 to 9 present the results for the HE and AHDE sequences for each of the basic heuristics when $M = 5$. Tables 10 to 13 present the results for the HE and AHDE sequences for each of the basic heuristics when $M = 10$. The results are averaged over all 100 instances in each set.

Using aggregation with the Webster, Jefferson, and bottleneck heuristics generates the best sequences. Compared to the HE sequences, the AHDE sequences reduce RTV dramatically when n is small or moderate. For instance, when $n = 300$ and $D = 1000$, the average RTV of the AHDE sequences is less than 10% of the average RTV of the HE sequences. When n is near D , the quality of the AHDE sequences is about the same as the quality of the HE sequences because the exchange heuristic is very good at constructing low-RTV sequences in those cases.

The sequential heuristic performs poorly overall. When n is small, the RTV of the HE and AHDE sequences exceeds the RTV of the HE and AHDE sequences generated using the other heuristics. When n is moderate, the RTV of the HE sequences is better (compared to the other HE sequences), but the RTV of the AHDE sequences is larger (compared to the other AHDE sequences). When n is large, the RTV of the HE and AHDE sequences is very low and close to the RTV of the HE and AHDE sequences generated using the other heuristics.

Table 14 lists the average lower bound for the instances in each set for both $M = 5$ and $M = 10$. When the number of machines is larger, the sequence length is shorter, which increases the RTV of a product because it is harder to spread out the production evenly. As n increases, the lower bounds decrease because the products have lower demand and are easier to distribute evenly. The RTV of the AHDE sequences generated using the Webster, Jefferson, and bottleneck heuristics is much closer to the lower bound than that of the HE sequences. For instance, when $M = 5$, $D = 1000$, and $n = 600$, the average RTV of the HE sequences generated using the Webster heuristic equals 100.4, the average RTV of the AHDE sequences equals 50.6, and the average lower bound equals 46.1, which shows that the AHDE sequences are near-optimal for these instances. For $M = 10$, the largest average gap between the lower bound and the RTV of the AHDE sequences generated using the Webster heuristic equals 9.7 when $D = 1500$ and $n = 200$. For this same set, the average gap between the lower bound and the RTV of the HE sequences generated using the Webster heuristic equals 1470.8.

The computational effort required to generate these sequences depended greatly upon D , the heuristic, and the type of sequence (see Figure 2). The computational effort increased significantly as D increased. For a given value of D , the following sequences took the most time: generating the H and HE sequences with the bottleneck heuristic (around 16 seconds per instance when $D = 1500$ and $M = 5$), generating the HE with the sequential heuristic (around 3 seconds when $D = 1500$ and $M = 5$), generating the HE sequences with the Webster and Jefferson heuristics (around 2 seconds when $D = 1500$ and $M = 5$), and generating the AHDE sequences with the sequential heuristic (just over 1 second when $D = 1500$ and $M = 5$). Generating the H sequences with the sequential heuristic took the least time by far, with average times around 0.0003 seconds when $D = 1500$ and $M = 5$.

For the Webster, Jefferson, and bottleneck heuristics, generating the AHDE sequences took dramatically less time than generating the HE sequences. For the Webster and Jefferson heuristics, with $M = 5$ and $D = 1500$, the average time needed to generate an AHDE sequence was less than 4% of the time needed to generate an HE sequence (0.0632 seconds to 1.6835 seconds). For the bottleneck heuristic, with $M = 5$ and $D = 1500$, the average time needed to generate an AHDE sequence was less than 1% of the time needed to generate an HE sequence (0.1524 seconds to 16.1263 seconds). Across all values of M , D , and n , generating AHDE sequences required less time with the Webster and Jefferson heuristics than with the bottleneck heuristic (using the Webster heuristic or Jefferson heuristic reduced the time by 15 to 80%).

In some cases, the computational effort required changed as n increased, but the magnitude of these changes was less significant than the changes due to changing other parameters. More time was required to construct the sequences with $M = 5$ than with $M = 10$ due to the longer sequence length. Generally, it took about twice as long to construct a sequence. None of the trends mentioned above changed with the increase in M .

Table 6. Comparison of the HE and AHDE sequences across four basic heuristics for $M = 5$ and $D = 100$.

n	Webster		Jefferson		Bottleneck		Sequential	
	HE	AHDE	HE	AHDE	HE	AHDE	HE	AHDE
40	47.7	8.5	46.1	8.4	53.4	8.5	20.8	16.2
50	15.6	7.2	17.7	7.0	19.6	7.2	9.3	9.4
60	5.5	4.8	8.0	4.8	7.8	4.9	6.6	6.0
70	3.4	2.9	3.3	3.1	3.6	3.0	3.5	3.4
80	1.5	1.3	1.3	1.3	1.3	1.4	1.3	1.6
90	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3

Table 7. Comparison of the HE and AHDE sequences across four basic heuristics
for $M = 5$ and $D = 500$.

n	Webster		Jefferson		Bottleneck		Sequential	
	HE	AHDE	HE	AHDE	HE	AHDE	HE	AHDE
50	364.7	88.6	357.6	90.4	360.1	90.9	540.9	437.8
100	472.3	73.5	470.0	72.8	460.9	75.0	209.6	246.7
150	603.8	52.2	564.9	52.5	552.7	55.4	349.5	308.9
200	281.2	46.1	264.6	45.7	336.6	46.0	140.2	138.6
250	89.7	36.0	99.1	36.1	160.1	37.7	57.5	55.7
300	40.4	25.7	47.3	25.5	128.8	26.9	34.7	32.1
350	17.4	14.7	15.9	14.8	16.9	15.6	16.9	15.7
400	8.0	6.5	8.0	6.4	7.0	6.5	6.7	7.5
450	1.6	1.5	1.6	1.5	1.5	1.5	1.5	1.5

Table 8. Comparison of the HE and AHDE sequences across four basic heuristics
for $M = 5$ and $D = 1000$.

n	Webster		Jefferson		Bottleneck		Sequential	
	HE	AHDE	HE	AHDE	HE	AHDE	HE	AHDE
50	577.1	206.5	574.2	211.8	579.6	206.6	3395.5	2885.6
100	1292.5	182.9	1291.2	185.8	1231.2	183.8	1150.8	993.0
200	1373.5	141.9	1176.1	141.7	1096.5	145.1	437.1	680.9
300	2961.6	106.0	1751.9	106.9	3077.1	110.6	994.1	1021.1
400	666.3	91.8	516.7	91.5	717.3	90.3	319.1	323.0
500	211.1	72.4	215.1	73.9	237.9	75.0	127.2	127.7
600	100.4	50.6	116.4	50.0	253.7	54.8	71.0	61.4
700	39.6	29.9	34.0	29.5	38.9	32.2	34.7	34.3
800	16.1	14.2	16.9	13.7	15.1	14.5	14.8	15.5
900	3.8	3.6	3.8	3.4	3.5	3.6	3.7	3.6

Table 9. Comparison of the HE and AHDE sequences across four basic heuristics
for $M = 5$ and $D = 1500$.

n	Webster		Jefferson		Bottleneck		Sequential	
	HE	AHDE	HE	AHDE	HE	AHDE	HE	AHDE
100	1887.0	297.9	1859.4	309.9	1873.2	307.3	3253.4	2695.6
200	2249.2	244.2	2113.6	250.9	2258.1	253.0	1288.0	1442.4
300	2352.8	179.8	1638.7	183.7	1727.7	184.0	779.9	1236.6
400	3928.6	132.9	3027.4	135.2	3332.9	147.9	2643.1	2434.6
500	2621.2	93.8	1374.1	93.0	1591.6	102.7	875.1	1151.3
600	930.0	56.4	691.4	56.5	1140.5	55.0	466.6	467.1
700	370.8	24.6	718.2	26.4	396.8	30.4	179.7	218.9
800	603.4	17.5	274.4	16.1	225.5	15.1	78.8	68.7
900	174.9	8.3	202.5	7.6	737.0	17.9	37.9	22.5
1000	81.4	4.3	46.3	3.2	154.6	9.7	14.9	15.8
1100	12.0	1.4	12.6	0.8	17.5	3.1	10.0	4.8
1200	3.4	0.7	4.0	0.4	5.0	1.8	1.5	2.3
1300	0.3	0.1	0.1	0.0	0.2	0.5	0.7	0.7
1400	0.2	0.0	0.2	0.0	0.0	0.1	0.1	0.1

Table 10. Comparison of the HE and AHDE sequences across four basic heuristics for $M = 10$ and $D = 100$.

n	Webster		Jefferson		Bottleneck		Sequential	
	HE	AHDE	HE	AHDE	HE	AHDE	HE	AHDE
40	32.7	13.1	28.7	12.9	24.1	12.9	15.6	14.2
50	20.0	10.1	18.5	9.7	20.5	9.7	10.2	10.4
60	7.8	6.4	8.0	6.3	11.9	6.4	6.6	6.6
70	3.5	3.5	3.5	3.5	3.5	3.5	3.5	3.5
80	1.5	1.5	1.5	1.4	1.4	1.4	1.5	1.5
90	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3

Table 11. Comparison of the HE and AHDE sequences across four basic heuristics for $M = 10$ and $D = 500$.

n	Webster		Jefferson		Bottleneck		Sequential	
	HE	AHDE	HE	AHDE	HE	AHDE	HE	AHDE
50	185.5	82.8	190.9	83.2	176.1	85.1	448.7	286.4
100	348.3	78.8	329.9	79.4	286.3	80.9	177.6	147.4
150	406.8	74.2	381.5	74.2	310.3	75.5	151.7	131.3
200	257.5	64.8	252.9	64.6	264.5	65.5	83.3	77.9
250	97.2	48.0	92.1	48.2	153.9	48.3	54.9	53.7
300	39.6	31.2	40.0	31.2	54.0	31.6	32.2	33.2
350	17.7	16.9	17.6	16.8	17.5	17.0	17.4	17.4
400	7.3	7.2	7.3	7.2	7.3	7.2	7.3	7.3
450	1.5	1.5	1.5	1.5	1.5	1.5	1.5	1.5

Table 12. Comparison of the HE and AHDE sequences across four basic heuristics for $M = 10$ and $D = 1000$.

n	Webster		Jefferson		Bottleneck		Sequential	
	HE	AHDE	HE	AHDE	HE	AHDE	HE	AHDE
50	312.2	169.0	313.2	168.7	290.4	171.2	2832.8	1563.7
100	712.6	149.9	691.4	150.8	698.5	155.0	955.8	648.7
200	916.0	130.6	890.2	131.1	893.9	135.1	341.8	348.0
300	1228.8	96.2	1090.4	94.7	1077.1	97.6	489.2	360.0
400	475.4	85.0	463.3	84.9	616.4	87.2	172.2	139.1
500	196.0	68.9	255.6	69.0	334.8	69.9	108.3	82.7
600	78.5	51.1	109.4	50.8	258.4	51.2	64.6	57.7
700	32.3	29.9	30.8	30.1	32.1	30.8	35.7	34.8
800	16.5	13.9	16.4	13.7	15.2	14.6	14.9	14.4
900	3.8	3.5	3.8	3.4	3.5	3.5	3.5	3.7

Table 13. Comparison of the HE and AHDE sequences across four basic heuristics for $M = 10$ and $D = 1500$.

N	Webster		Jefferson		Bottleneck		Sequential	
	HE	AHDE	HE	AHDE	HE	AHDE	HE	AHDE
100	911.3	265.6	926.8	266.4	938.3	273.1	2752.0	1952.5
200	1696.3	235.2	1655.5	237.3	1654.2	241.3	931.8	795.2
300	1685.8	172.2	1669.0	173.3	1698.9	176.2	546.0	662.1
400	1597.5	134.4	1557.6	134.1	1733.5	137.7	863.0	655.9
500	2957.2	111.8	1337.6	111.8	1765.3	113.3	473.6	465.4
600	742.3	85.4	892.6	85.0	915.5	88.1	209.7	173.0
700	335.1	62.6	686.9	63.3	378.1	63.0	135.9	114.6
800	160.6	40.6	289.1	40.8	539.5	44.3	67.9	58.2
900	144.2	24.6	183.8	24.3	928.6	26.2	27.5	31.5
1000	78.7	14.2	52.1	13.6	120.3	15.0	14.9	16.4
1100	11.0	6.6	10.4	6.2	9.7	6.7	6.7	7.2
1200	6.0	2.6	7.2	2.6	3.9	2.7	2.8	2.8
1300	0.7	0.7	0.7	0.6	0.7	0.6	0.7	0.7
1400	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

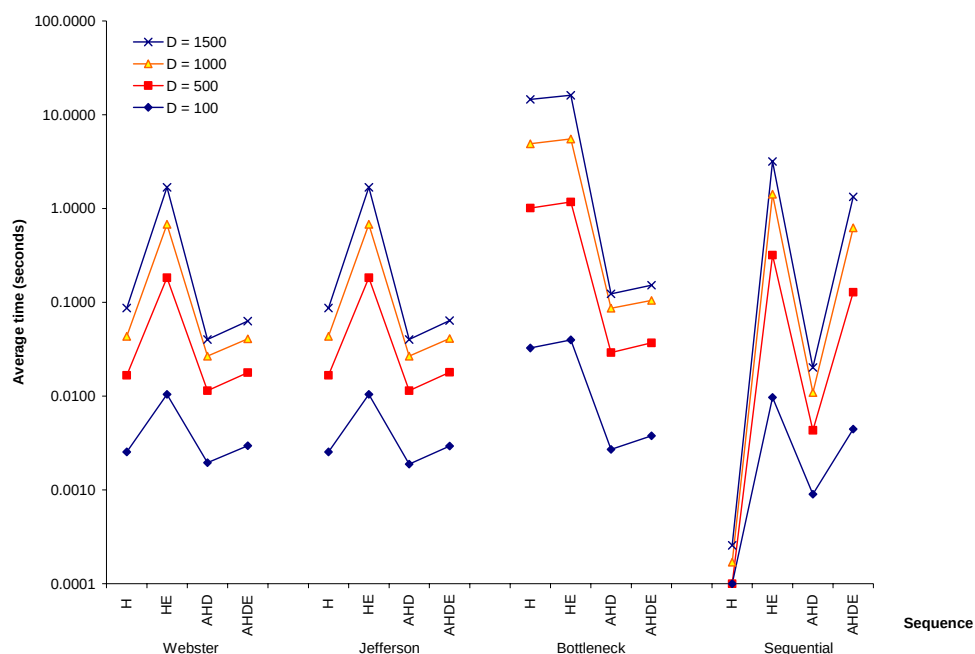


Figure 2. Average time required to generate different sequences using different heuristics when $M = 5$. Results are averaged over all instances with the given value of D .
Note that the vertical axis is logarithmic to improve readability.

Table 14. Average Lower Bound.

D	n	Lower Bound	
		$M = 5$	$M = 10$
100	40	7.5	12.7
	50	6.5	9.5
	60	4.6	6.3
	70	2.8	3.5
	80	1.3	1.4
	90	0.3	0.3
500	50	72.9	81.6
	100	61.2	77.0
	150	44.0	72.9
	200	39.1	63.7
	250	31.2	46.9
	300	23.4	30.8
	350	14.1	16.7
	400	6.4	7.2
	450	1.5	1.5
1000	50	162.3	161.8
	100	149.8	144.8
	200	111.4	124.1
	300	86.2	88.5
	400	77.6	77.8
	500	63.9	63.9
	600	46.1	46.1
	700	28.2	28.2
	800	13.4	13.4
	900	3.4	3.4
1500	100	210.8	258.2
	200	169.6	225.5
	300	84.7	162.6
	400	29.9	126.8
	500	9.5	103.9
	600	2.9	79.3
	700	0.9	56.5
	800	0.2	37.3
	900	0.1	22.2
	1000	0.0	13.1
	1100	0.0	6.0
	1200	0.0	2.5
	1300	0.0	0.6
	1400	0.0	0.1

7 Summary and Conclusions

This paper presents an aggregation approach for the problem of minimizing RTV when there are multiple servers to process the products. We combined this approach with various heuristics for the multiple-server RTV problem in order to determine when aggregation is useful.

The results show that, in most cases, combining aggregation with other heuristics does dramatically reduce both RTV and computation time compared to using the heuristics without

aggregation (when the exchange heuristic is used to improve the solutions found). In these cases, the solutions generated by the heuristics have large values of RTV, and aggregation provides a way to find much better solutions. Aggregation is particularly effective in combination with the Webster, Jefferson, and bottleneck heuristics. Comparing the RTV of the sequences generated using aggregation to a lower bound shows that the sequences are near-optimal solutions.

We can see why aggregation improves RTV if we consider a set of products with the same demand. The Webster, Jefferson, and bottleneck heuristics tend to put copies of these products next to each other, which increases the RTV of products with higher demand. Because it combines them into one group, aggregation spreads out the products with the same demand, leaving space for the products with higher demand.

Due to the larger computational effort of the bottleneck heuristic, using aggregation with the Webster and Jefferson heuristics (instances of the parameterized stride scheduling heuristic) will be the most effective approach. We note that Corominas et al. [11] found that, without aggregation, the Webster and Jefferson heuristics generated single-server sequences with higher RTV than those generated by the bottleneck heuristic.

On the other hand, when the number of products is near the total demand (and each product has very small demand), combining aggregation with other heuristics does not reduce RTV compared to using the heuristics without aggregation (when the exchange heuristic is used to improve the solutions found). Still, in these cases, aggregation does reduce the computational effort needed to generate sequences.

The aggregation procedure described in this paper is simple but effective. However, it is possible to create more sophisticated aggregations that more intelligently combine products in order to minimize the number of products in the highest level of aggregation, with the goal of aggregating the products into one group for each server. If there is only one group for each server, the disaggregation leads directly to a zero RTV sequence. It will be useful to consider the use of optimization approaches (which can solve only small RTV problems) on the aggregated instances to determine if aggregation can reduce RTV and computational effort with that approach as well.

Acknowledgements This work was motivated by a collaboration with the University of Maryland Medical Center. The author appreciates the help of Leonard Taylor, who introduced the problem, provided useful data, and recognized the value of the work presented here.

References

1. Kubiak, W., Fair sequences, in Handbook of Scheduling: Algorithms, Models and Performance Analysis, Leung, J.Y-T., editor, Chapman & Hall/CRC, Boca Raton, Florida, 2004.
2. Bar-Noy, Amotz, Randeep Bhatia, Joseph Naor, and Baruch Schieber, Minimizing service an operation costs of periodic scheduling, Mathematics of Operations Research, 27, 518-544, 2002.
3. Bar-Noy, Amotz, Aviv Nisgah, and Boaz Patt-Shamir, Nearly Optimal Perfectly Periodic Schedules, Distributed Computing, 15, 207-220, 2002.
4. Wei, W.D., and Liu, C.L., On a Periodic Maintenance Problem, Operations Research Letters, 2, 90-93, 1983.
5. Park, Kyung S., and Doek K. Yun, Optimal Scheduling of Periodic Activities, Operations Research, 33, 690-695, 1985.
6. Campbell, Ann Melissa, and Jill R. Hardin, Vehicle Minimization for Periodic Deliveries, European Journal of Operational Research, 165, 668-684, 2005.
7. Waldspurger, C.A., and Weihl, W.E., Stride Scheduling: Deterministic Proportional-Share Resource Management, Technical Memorandum MIT/LCS/TM-528, MIT Laboratory for Computer Science, Cambridge, Massachusetts, 1995.

8. Balinski, M.L., and H.P. Young, Fair Representation, Yale University Press, New Haven, Connecticut, 1982.
9. Miltenburg, J., Level Schedules for Mixed-Model Assembly Lines in Just-in-Time Production Systems, *Management Science*, 35, 192-207, 1989.
10. Inman, R.R., and Bulfin, R.L., Sequencing JIT Mixed-Model Assembly Lines, *Management Science*, 37, 901-904, 1991.
11. Corominas, Albert, Wieslaw Kubiak, and Natalia Moreno Palli, Response Time Variability, *Journal of Scheduling*, 10, 97-110, 2007.
12. Garcia, A., R. Pastor, and A. Corominas, Solving the Response Time Variability Problem by Means of Metaheuristics, in *Artificial Intelligence Research and Development*, edited by Monique Polit, T. Talbert, and B. Lopez, pages 187-194, IOS Press, 2006.
13. Kubiak, Wieslaw, *Proportional Optimization and Fairness*, Springer, New York, 2009.
14. Herrmann, Jeffrey W., Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling, Technical Report 2007-12, Institute for Systems Research, University of Maryland, College Park, 2007.
15. Herrmann, Jeffrey W., Using Aggregation to Reduce Response Time Variability in Cyclic Fair Sequences, Technical Report 2008-20, Institute for Systems Research, University of Maryland, August, 2008.
16. Herrmann, Jeffrey W., Constructing Perfect Aggregations to Eliminate Response Time Variability in Cyclic Fair Sequences, Technical Report 2008-29, Institute for Systems Research, University of Maryland, October, 2008.
17. Rogers, David F., Robert D. Plante, Richard T. Wong, and James R. Evans, Aggregation and Disaggregation Techniques and Methodology in Optimization, *Operations Research*, 39, 553-582, 1991.
18. Steiner, George, and Scott Yeomans, Level Schedules for Mixed-Model, Just-in-Time Processes, *Management Science*, 39, 728-735, 1993.

Scheduling two interfering job sets on uniform parallel machines with makespan and cost functions

Donatas Elvikis · Horst W. Hamacher ·
Vincent T'kindt

Abstract We consider a m uniform parallel machines scheduling problem with two jobs A and B with n_A and n_B operations, respectively, and equal processing times. Job A is associated with a general cost function F^A , whereas makespan is considered for job B . As we are dealing with two conflicting criteria, we focus on the calculation of strict Pareto optima for F^A and C_{max}^B criteria. We also tackle some particular cases of F^A and provide polynomial algorithms.

1 Introduction

In this paper we consider m uniform parallel machines with processing speeds denoted by v_j . Two jobs A and B , each consisting of n_A and n_B operations, respectively, have to be scheduled. We denote by $\mathcal{I} = A \cup B$ the set of all operations and $n = |\mathcal{I}| = n_A + n_B$ its cardinality. Each operation i requires a processing time p_i which is assumed to be equal for all operations, *i.e.* $\forall i \in \mathcal{I}, p_i = p$. Notice that further in the paper we denoted by $p_j = \frac{p}{v_j}$ a processing time of any operation scheduled on machine j .

To job A we associate a general cost function, referred to as F^A , which is assumed to be minimized. More precisely, we consider in this paper two cases for F^A : either $F_{\Sigma}^A = \sum_{i \in A} f_i(C_i)$ or $F_{max}^A = \max_{i \in A} f_i(C_i)$, with C_i the completion time of operation i and f_i an increasing function. Similarly, we associate to job B criterion C_{max}^B , defined by $C_{max}^B = \max_{i \in B} (C_i)$, which is to be minimized. Henceforth, we faced with a two interfering job sets problem, *i.e.* scheduling each job is done according to its own cost function while sharing common resources with the other job. It is obvious that jobs are selfish and criteria are conflicting, which means we have a bicriteria scheduling

Donatas Elvikis
University of Kaiserslautern, Germany
E-mail: elvikis@mathematik.uni-kl.de

Horst W. Hamacher
University of Kaiserslautern, Germany
E-mail: hamacher@mathematik.uni-kl.de

Vincent T'kindt
LI, Université de Tours, France
E-mail: vincent.tkindt@univ-tours.fr

problem. This one is referred to as $Q|p_i = p|\#(F^A, C_{max}^B)$ according to the notation introduced in [13] and extended in [15].

As usual when dealing with multiple conflicting criteria, we focus on the calculation of strict Pareto optima for F^A and C_{max}^B criteria. A schedule σ is a strict Pareto optimum if there does not exist another schedule σ' such that $F^A(\sigma') \leq F^A(\sigma)$ and $C_{max}^B(\sigma') \leq C_{max}^B(\sigma)$ with at least one strict inequality. Such optima have the property to be also weak Pareto optima: a schedule σ is a weak Pareto optimum if there does not exist another schedule σ' such that $F^A(\sigma') < F^A(\sigma)$ and $C_{max}^B(\sigma') < C_{max}^B(\sigma)$. Notice that a weak Pareto optimum may not be a strict one. At last, to each strict Pareto optimum is associated a strictly non-dominated criteria vector (F^A, C_{max}^B) .

Scheduling interfering job sets is a quite recent research area. First, Agnetis et al. ([1]) introduce a jobshop scheduling problem with two competing players each one having its own optimization goal: this was the starting point for interfering job sets problems. For the tackled jobshop problem they focus on the enumeration of Pareto optima for two general quasiconvex functions of the operation completion times.

Baker and Smith ([3]) study several simple single machine problems with two interfering job sets where scheduling is done by minimizing a convex combination of two criteria among C_{max} , L_{max} and $\sum_i w_i C_i$. They provide polynomial time algorithms for calculating the optimal solution (a Pareto optimum) for problems $1|d_i|F_\ell(C_{max}^A, L_{max}^B)$ and $1|d_i|F_\ell(C_{max}^A, \sum_i w_i C_i^B)$. They also prove that problem $1|d_i|F_\ell(L_{max}^A, \sum_i w_i C_i^B)$ is $\mathcal{NP}$ -hard except if $w_i = 1, \forall i = 1, \dots, n$. Later on, Yuan et al. ([17]) propose an update for some of Baker and Smith's results, notably for the $1|d_i|F_\ell(L_{max}^A, L_{max}^B)$ and $1|d_i|F_\ell(\sum_i C_i^A, L_{max}^B)$ problems for which they propose a dynamic program requiring $O(n_A n_B (n_A + n_B))$ time. Agnetis et al. ([2]) tackle a bunch of single machine scheduling problems among which are the $1||\varepsilon(\sum_i w_i C_i^A / F_{max}^B)$ and $1||\varepsilon(F_{max}^A / F_{max}^B)$ problems. They show that the former is $\mathcal{NP}$ -hard (by reduction from the knapsack problem) except if $w_i = 1, \forall i = 1, \dots, n_A$, and the latter can be solved in $O(n_A^2 + n_B \log(n_B))$ time. They also focus on the cardinality of the set of strict Pareto optima for the $1||\#(F_{max}^A, F_{max}^B)$ problem and show that this set contains at most $n_A n_B$ solutions (in criteria space). Agnetis et al. also consider two 2-machine shop problems which are shown to be $\mathcal{NP}$ -hard. The first problem with more than two interfering job sets is dealt with by Cheng et al. ([10]) who consider the $1|d_i, \sum_i w_i U_i^{(\ell)} \leq \varepsilon^{(\ell)}|$, i.e. for each interfering job we impose a bound constraint on its weighted number of tardy operations. They show that this problem is strongly $\mathcal{NP}$ -hard.

To the best of our knowledge, no work deals with parallel machine problems.

One other feature of the problems tackled in this paper is that all operations are of equal size. This case has been the matter of numerous studies along the last decade. Regarding single machine problems, Baptiste studies the $1|r_i, p_i = p, |\sum w_i U_i$ problem ([4]) and the $1|r_i, p_i = p, |\sum T_i$ problem ([5]) for which he proposes polynomial time algorithms. Later on, Chrobak et al. ([11]) consider the $1|r_i, p_i = p, |\sum U_i$ problem for which they propose a modification of Baptiste's algorithm with improved time complexity. Other single machine problems can be found in [7].

Parallel machine problems have also been tackled, mainly when machines are identical. Baptiste ([5]) focus on the two $Pm|r_i, p_i = p|\sum w_i C_i$ and $Pm|r_i, p_i = p|\sum T_i$ problems and show that they can be solved in polynomial time. Later on ([6]), he shows that the case for which each job requires a given number of machines the problem, referred to as $Pm|r_i, p_i = p, size_i|\sum C_i$, is still polynomially solvable. Brucker and Kravchenko ([9]) focus on some problems among which the $P|r_i, p_i = p|\sum w_i C_i$ problem that is shown to be solvable in polynomial time. Other problems, notably

some with preemption allowed, can be found in [7], [8] and [9]. We now turn to some works more related to the problems tackled in this paper.

Dessouky et al. ([12]) study a set of simple but important scheduling problems on uniform parallel machines with unit processing time jobs among which the $Q|p_i = 1|F_{\Sigma}$ and $Q|p_i = 1|F_{\max}$ problems which can be solved in $O(n^3)$ and $O(n^2)$ time, respectively. The first one is solved by means of an assignment problem whilst the second one is solved by a simple constructive algorithm. They also provide specializations to particular classic max and sum scheduling criteria.

Another important contribution is due to Tuzikov et al. who develops ([16]) two polynomial time algorithms for $Q|p_i = p|\varepsilon(F_{\max}^1/F_{\max}^2)$ and $Q|p_i = p|\varepsilon(F_{\Sigma}^1/F_{\max}^2)$ problems which are then iteratively solved to calculate the strict Pareto set. Even though these problems may seem similar to the one tackled in this paper, they are thoroughly different since we consider two interfering job sets. Moreover, they show that the size of the Pareto set is bounded by at most n^2 solutions for both problems, while we refine this bound for our specific problem and show that there are no more than $n_A + 1$ non-dominated points in the criteria space.

The remainder is organized as follows. We first focus, in Section 2, on general results about equal-size operations and minimization of criterion $C_{\max}^B$. In Section 3 we consider the particular case of F_{Σ}^A and $C_{\max}^B$ criteria and provide a polynomial time algorithm for enumerating the set of strictly non-dominated criteria vectors. In Section 4 we consider the case $F_{\max}^A$ and $C_{\max}^B$ criteria and, again, provide a polynomial time algorithm for enumerating the set of strictly non-dominated criteria vectors.

2 General properties of the $Q|p_i = p|\#(F^A, C_{\max}^B)$ problem

In this section we focus on some general properties of the $Q|p_i = p|\#(F^A, C_{\max}^B)$ problem, whatever F^A . First, observe that as far as criterion F^A is a regular one, *i.e.* an increasing function of the completion times, we only need to restrict the search for Pareto optima in the set of active schedules (see [12] or [15] for instance). Consequently, and due to the fact that operations are of equal-size, the completion times of operations scheduled on a given machine j , $\forall j = 1, \dots, m$, are equal to $kp_j = k\frac{p}{v_j}$ for any position k . Let us define $\mathcal{T}_j$ as the set completion of times for machine j by

$$\mathcal{T}_j = \{t \mid t = kp_j, k \in \mathbb{N}\}. \quad (1)$$

Assume that all timeslots in $\bigcup_{j=1, \dots, m} \mathcal{T}_j$ are numbered in ascending order of their completion times and, in case of ties, slot on a faster machine gets lower index, *i.e.* $\mathcal{T} = \{t_1 \leq t_2 \leq \dots \leq t_n\} \subseteq \bigcup_{j=1, \dots, m} \mathcal{T}_j$. Therefore $\mathcal{T}$ is the set of completion times for any active schedule σ and minimizing criteria F^A and $C_{\max}^B$ is equivalent to assign operations to completions times. In other words, the uniform parallel machines problem can be seen as a particular single machine problem for which the sequence of completion times is fixed and we only need to assign jobs to completion times which makes it more particular than the problems tackled by Agnetis et al. ([2]).

We now turn to the situation in which $C_{\max}^B \leq \varepsilon$, where ε is given, and F^A is minimized. This ε -constrained problem, referred to as $Q|p_i = p|\varepsilon(F^A/C_{\max}^B)$, will be solved later on to compute a Pareto optimum. Assume that the last operation of job B completes at timeslot t , then it is obvious that the assignment of the other operations of B to timeslots $t_j < t$ does not matter. We now provide a similar result than the Property 1 of Baker and Smith ([3]) for the single machine case.

Lemma 1 *There exists an optimal schedule σ to the $Q|p_i = p|\varepsilon(F^A/C_{max}^B)$ problem such that all operations of job B are sequenced in a single block completing at time C_{max}^B , i.e. no operation of job A is scheduled between two operations of job B .*

Proof Assume that σ does not satisfies this property, i.e. there exists a timeslot k such that an operation i of job A completes at time $t_k \leq C_{max}^B$ and some operation j of job B completes at time t_{k-1} . By swapping j and i we obtain a schedule for which C_{max}^B is unchanged and, since F^A is a regular criterion its value is not increased. By repeatedly applying such swaps on an optimal schedule we build another optimal schedule for which all operations of B form a block. $\square$

Let us denote by n_ε the number of timeslots which complete before or at ε , thus we have

$$n_\varepsilon = \left| \left\{ t_k \in \bigcup_{j=1, \dots, m} \mathcal{T}_j \mid t_k \leq \varepsilon \right\} \right|. \quad (2)$$

Besides, as F^A is regular and due to Lemma 1, minimizing F^A is equivalent to finding an optimal assignment of job A operations to the $t_1, \dots, t_{n_\varepsilon - n_B}, t_{n_\varepsilon + 1}, \dots, t_n$ timeslots. This can be basically done in $O(n_A^2)$ or $O(n_A^3)$ time by applying the algorithms presented in [12].

Lemma 2 *There are at most $n_A + 1$ strict Pareto optima in criteria space.*

Proof This follows directly from the fact that there are at most $n_A + 1$ possible completion times for job B . $\square$

The above general solution procedure can be applied iteratively with different ε values in order to enumerate the set $\mathcal{P}$ of strict Pareto optima (see Algorithm 1). This algorithm can be implemented in $O(n_B \log(m) + n_A(T + \log(m)))$ time, with T the time complexity for optimally assigning operations of job A (Step 7).

Algorithm 1: Solution of the $Q|p_i = p|\#(F^A, C_{max}^B)$ problem.

Data: An instance of the $Q|p_i = p|\#(F^A, C_{max}^B)$ problem.
Result: Strict Pareto set $\mathcal{P}$.

```

1 begin
2    $\mathcal{P} \leftarrow \emptyset;$                                      /* Initialize Pareto set. */
3    $\varepsilon \leftarrow t_{n_B}, \quad n_\varepsilon \leftarrow n_B;$       /* Set initial  $\varepsilon$  and  $n_\varepsilon$  values. */
4   repeat
5      $n_\varepsilon \leftarrow \operatorname{argmax}_{k=n_\varepsilon, \dots, n} \{t_k \leq \varepsilon\};$  /* Index of the last feasible timeslot
6       for the operation in job  $B$ . */
7     Put the whole job  $B$  into  $t_{n_\varepsilon - n_B + 1}, \dots, t_{n_\varepsilon}$  timeslots;
7     Sequence operations of  $A$  to the  $t_1, \dots, t_{n_\varepsilon - n_B}, t_{n_\varepsilon + 1}, \dots, t_n$  timeslots;
8      $\mathcal{P} \leftarrow \mathcal{P} \cup \{(\sigma_\varepsilon, F^A, C_{max}^B = \varepsilon)\};$  /* Append solution to the Pareto set. */
9      $\varepsilon \leftarrow t_{n_\varepsilon + 1};$                           /* Increase  $\varepsilon$  value. */
10  until  $\exists \varepsilon$ ;
11  Remove weakly non-dominated solutions from  $\mathcal{P}$ ;
12  return  $\mathcal{P}$ ;
13 end
```

3 Solving the $Q|p_i = p|\#(F_{\Sigma}^A, C_{max}^B)$ problem

In this section we focus on the case where $F^A = F_{\Sigma}^A = \sum_{i \in A} f_i$, with f_i 's being increasing functions of the completion times. First, observe that sequencing job A in Step 7 of Algorithm 1 can be basically achieved by solving an assignment problem in $O(n_A^3)$ time, thus leading to an overall $O((n_A + n_B) \log(m) + n_A^4)$ time complexity for Algorithm 1. We show that this complexity can be reduced if the Hypothesis 1 holds.

Hypothesis 1 Let $f_{i'}$ and $f_{i''}$ be two non-decreasing functions: $\forall t_2 \geq t_1 \geq 0$, we have either $(f_{i'} - f_{i''})(t_1) \leq (f_{i'} - f_{i''})(t_2)$ or $(f_{i'} - f_{i''})(t_1) \geq (f_{i'} - f_{i''})(t_2)$, with $(f_{i'} - f_{i''})(t) = f_{i'}(t) - f_{i''}(t)$.

It is interesting to notice that some of the classic sum criteria in scheduling answer this hypothesis. For instance, this is the case of the weighted sum of completion times criterion $\sum_i w_i C_i$ for which $f_i(t) = w_i t$. This hypothesis also holds for the total tardiness criterion $\sum_i T_i$, with $f_i(t) = \max(0, t - d_i)$ and for the total weighted tardiness $\sum_i w_i T_i$ if weights w_i and due dates d_i are agreeable, i.e. $d_{i'} \leq d_{i''} \Rightarrow w_{i'} \geq w_{i''}$.

In the remainder of this section we assume that Hypothesis 1 holds and we make use of the following instrumental notations: $\sigma(k)$ denotes operation sequenced in position k in sequence σ and $t_k \in \mathcal{T}$ is its completion time, i.e. the completion time of the timeslot in which $\sigma(k)$ is scheduled. We are now ready to present an important result which states that the optimal sequence of job A operations does not depend on the value of the timeslots.

Lemma 3 Assume that we have two sets of timeslots $\mathcal{T}' = \{t'_1 \leq \dots \leq t'_n\}$ and $\mathcal{T}'' = \{t''_1 \leq \dots \leq t''_n\}$ such that $\mathcal{T}' \neq \mathcal{T}''$. Let σ' (resp. σ'') be the optimal sequence of operations for criterion F_{Σ} with respect to the timeslots of $\mathcal{T}'$ (resp. $\mathcal{T}''$). We have $\sigma' = \sigma''$.

Proof Let $k < n$ be the index such that $\sigma'(v) = \sigma''(v)$, $\forall v = 1, \dots, k-1$, and $\sigma'(k) \neq \sigma''(k)$. We assume that σ' and σ'' are maximally equal, i.e. there does not exist other optimal sequences σ' and σ'' with a higher k value.

Consider operation $i' = \sigma'(k)$ which is scheduled in σ'' in position $u > k$, and let us denote by $i'' = \sigma''(u-1)$. Besides, we denote by ℓ the index such that $\sigma'(\ell) = i''$ and we have $\ell > k$.

As σ' is optimal regarding timeslots $\mathcal{T}'$, swapping i' and i'' in σ' we obtain:

$$\delta'_{\Sigma} = F_{\Sigma}(\sigma'(i' \leftrightarrow i'')) - F_{\Sigma}(\sigma') \geq 0,$$

where $\sigma'(i' \leftrightarrow i'')$ the sequence σ' with operations i' and i'' swapped. As σ' is assumed to be optimal for timeslots $\mathcal{T}'$, we have:

$$\begin{aligned} \delta'_{\Sigma} &= (f_{i'}(t_{\ell}) + f_{i''}(t_k)) - (f_{i'}(t_k) + f_{i''}(t_{\ell})) \\ &= (f_{i'} - f_{i''})(t_{\ell}) - (f_{i'} - f_{i''})(t_k) \geq 0. \end{aligned}$$

Recall that $t_{\ell} \geq t_k$, thus function $(f_{i'} - f_{i''})(t)$ is a non-decreasing function. Now, let us turn to σ'' which is optimal regarding timeslots $\mathcal{T}''$, swapping i' and i'' we obtain:

$$\begin{aligned} \delta''_{\Sigma} &= F_{\Sigma}(\sigma''(i' \leftrightarrow i'')) - F_{\Sigma}(\sigma'') \\ &= (f_{i''}(t_u) + f_{i'}(t_{u-1})) - (f_{i''}(t_{u-1}) + f_{i'}(t_u)) \\ &= (f_{i'} - f_{i''})(t_{u-1}) - (f_{i'} - f_{i''})(t_u). \end{aligned}$$

As $(f_{i'} - f_{i''})(t)$ is a non-decreasing function and $t_u \geq t_{u-1}$ we deduce that $\delta''_{\Sigma} \leq 0$. But as σ'' is assumed to be optimal regarding timeslots $\mathcal{T}''$, we necessarily have $\delta''_{\Sigma} = 0$ and swapping i' and i'' does not increase the objective function value of F_{Σ} over $\mathcal{T}''$. But this clearly contradict the fact that σ' and σ'' have been chosen as maximally equal. Consequently, we necessarily have $\sigma' = \sigma''$. $\square$

Considering Lemma 3 we can improve Algorithm 1 to reduce its time complexity. Observe that in the resulting Algorithm 2, the $Q|p_i = p|F_{\Sigma}^A$ problem is solved only once, to produce the optimal sequence σ^* , in Step 4. Then, in Step 8 the operations of job A are assigned to the free timeslots in their order in sequence σ^* . The overall complexity for finding all non-dominated criteria vectors is in $\mathcal{O}((n_A + n_B) \log(m) + T)$ time, with T the time complexity needed to solve the $Q|p_i = p|F_{\Sigma}^A$ problem. Notice that if we want to save one schedule per criteria vector, we need to pay an extra cost which yields in this case a complexity in $\mathcal{O}(n_B \log(m) + T + n_A(n + \log(m)))$ time.

Algorithm 2: Solution of the $Q|p_i = p|\#(F_{\Sigma}^A, C_{max}^B)$ problem under Hypothesis 1.

Data: An instance of the $Q|p_i = p|\#(F_{\Sigma}^A, C_{max}^B)$ problem.
Result: Strict Pareto set $\mathcal{P}$.

```

1 begin
2    $\mathcal{P} \leftarrow \emptyset$ ; /* Initialize Pareto set. */
3    $\varepsilon \leftarrow t_{n_B}, \quad n_{\varepsilon} \leftarrow n_B$ ; /* Set initial  $\varepsilon$  and  $n_{\varepsilon}$  values. */
4   Let  $\sigma^*$  be an optimal solution of the  $Q|p_i = p|F_{\Sigma}^A$  problem;
5   repeat
6      $n_{\varepsilon} \leftarrow \operatorname{argmax}_{k=n_{\varepsilon}, \dots, n} \{t_k \leq \varepsilon\}$ ; /* Index of the last feasible timeslot
7     for the operation in job  $B$ . */
8     Put the whole job  $B$  into  $t_{n_{\varepsilon}-n_B+1}, \dots, t_{n_{\varepsilon}}$  timeslots;
9     Sequence operations of  $A$  to the  $t_1, \dots, t_{n_{\varepsilon}-n_B}, t_{n_{\varepsilon}+1}, \dots, t_n$  timeslots
10    following sequence  $\sigma^*$ ;
11     $\mathcal{P} \leftarrow \mathcal{P} \cup \{(\sigma_{\varepsilon}, F_{\Sigma}^A, C_{max}^B = \varepsilon)\}$ ; /* Append solution to the Pareto set. */
12     $\varepsilon \leftarrow t_{n_{\varepsilon}+1}$ ; /* Increase  $\varepsilon$  value. */
13  until  $\exists \varepsilon$ ;
14  Remove weakly non-dominated solutions from  $\mathcal{P}$ ;
15  return  $\mathcal{P}$ ;
16 end
```

We now turn to special cases of the F_{Σ} cost function for which Hypothesis 1 holds, and we first start with $F_{\Sigma}^A = \sum_i w_i C_i^A$, $w_i \geq 0, \forall i = 1, \dots, n_A$. It is well known ([12]) that the $Q|p_i = p|\sum_i w_i C_i$ can be solved in polynomial time by a straight adaptation of Smith's rule: schedule at any time the job with the greatest weight w_i to the machine which completes it the earliest. Henceforth, in Step 4 of Algorithm 2 sequence σ^* is obtained by sorting operations i of job A by decreasing value of their weight. Therefore, the $Q|p_i = p|\#(\sum_i w_i C_i^A, C_{max}^B)$ problem can be solved in $\mathcal{O}((n_A + n_B) \log(m) + n_A \log(n_A))$ time.

Additionally, both $Q|p_i = p|F_{\Sigma}^A$ problems with $F_{\Sigma}^A = \sum_i T_i^A$ and $F_{\Sigma}^A = \sum_i w_i T_i^A$ (with agreeable weights and due dates) can be solved ([12])) by a straight adaptation of Jackson's rule: schedule at any time the job with the smallest due date d_i to the machine which completes it the earlier. Again, sequence σ^* is obtained by sorting

operations of job A by increasing value of their due date. This also results in overall $\mathcal{O}((n_A + n_B) \log(m) + n_A \log(n_A))$ time complexity for Algorithm 2.

4 Solving the $Q|p_i = p|\#(F_{max}^A, C_{max}^B)$ problem

In this section we focus on the case where $F^A = F_{max}^A = \max_{i \in A} f_i$ with the f_i 's being increasing functions of the completion times. This type of criterion has been often considered in the literature since it generalizes classic scheduling criterion as the makespan, the maximum tardiness and the maximum lateness criteria. A straightforward implementation of Algorithm 1 can be derived for the $Q|p_i = p|\#(F_{max}^A, C_{max}^B)$ problem by using Lawler's algorithm ([14]) when applying Step 7. Consequently, this step can be solved in $\mathcal{O}(n_A^2)$ time and the overall complexity of the enumeration algorithm becomes in $\mathcal{O}((n_A + n_B) \log(m) + n_A^3)$ time. In the remainder of this section, we focus on two particular and widely used scheduling criteria, namely the makespan $C_{max}^A = \max_{i \in A} C_i$ and the maximum lateness $L_{max}^A = \max_{i \in A} (C_i - d_i)$.

First it is obvious that $Q|p_i = p|\#(C_{max}^A, C_{max}^B)$ problem has only two strictly non-dominated solutions. One with $(C_{max}^A = t_{n_A}, C_{max}^B = t_n)$ and the other with $(C_{max}^A = t_n, C_{max}^B = t_{n_B})$ criteria vectors.

Now, we turn to the maximum lateness criterion. Even though Lemma 3 in general does not apply for F_{max}^A functions we show that in case of L_{max}^A a similar result holds.

Lemma 4 *Whatever the timeslots $\mathcal{T} = \{t_1 \leq \dots \leq t_n\}$ are, the $Q|p_i = p|L_{max}^A$ problem can be solved by sorting operations of job A in ascending order of their due dates (EDD rule) and assigning them in this order to the timeslots.*

Proof This result can be shown by a simple interchange argument. $\square$

The use of the above lemma inside Algorithm 2 leads to an overall $\mathcal{O}((n_A + n_B) \log(m) + n_A \log(n_A))$ time complexity to find non-dominated criteria vectors and $\mathcal{O}((n_A + n_B) \log(m) + n_A n)$ if corresponding schedules are of interest. However it is still possible to reduce the practical time complexity of the enumeration algorithm by proposing one which eliminates from computation the weakly non-dominated criteria vectors. The idea is to avoid, for each ε value, to build from scratch the non-dominated solutions.

First let us assume that operations in A are divided into two sets A' and A'' , where A' contains all operations that are sequenced before job B in a current schedule σ and A'' is then defined by $A'' = A \setminus A'$. The first result below introduces a stopping criteria for the algorithm.

Lemma 5 *If schedule σ' is a non-dominated solution for the $Q|p_i = p|(L_{max}^A, C_{max}^B)$ problem with criteria vector $(L_{max}^{'A}, C_{max}^{'B})$ and the $L_{max}^{'A}$ value is given by an operation of set A' , then there is no other non-dominated schedule σ'' with $L_{max}^{'A} < L_{max}^{'A}$.*

Proof To decrease the $L_{max}^{'A}$ value we have to increase $C_{max}^{'B}$, i.e. we right timeshift the block of operations of job B . Consequently, some operations of set A'' are moved to the set A' but, due to Lemma 4 the operations in A' still follows the EDD order and we have not decreased the $L_{max}^{'A}$ value. $\square$

From Lemma 5 we can derive that Algorithm 2, which starts by scheduling first operations of job B , must stop when the L_{max}^A value is given by an operation sequenced in set A' . At each iteration a new ε value is set and the operations of job B are therefore right timeshifted. However not all ε values lead to strictly non-dominated solutions. Lemma 6 states a condition on how much operations of job B must be right timeshifted in order to get a new strictly non-dominated solution.

Lemma 6 *Let σ' be a strictly non-dominated solution with criteria vector (L_{max}^A, C_{max}^B) , with the L_{max}^A value given by the operation $i \in A''$ (break ties by choosing the latest operation in A'' which gives the L_{max}^A value). The next strictly non-dominated schedule σ'' with $L_{max}^A < L_{max}^A$ is obtained by moving from set A'' to set A' all operations scheduled before operation i (including i).*

Proof Given a schedule σ'' such that $L_{max}^A < L_{max}^A$ implies that operation $i \in A$ completes earlier than in σ' . This however means that i was swapped with at least one operation $u \in \mathcal{I}$ sequenced before i in σ' . First suppose that $u \in A$, however u is scheduled before i in σ' , thus due to Lemma 4 inequality $d_u \leq d_i$ holds, which implies two cases

- (i) either $d_u = d_i$ with $L_{max}^A = L_{max}^A$
- (ii) or $d_u < d_i$ with $L_{max}^A > L_{max}^A$

which both contradict to the assumption above, hence operation u must belong to job B . Furthermore, due to Lemma 1 operations in B can be considered as a single block, thus i and all operations $j \in A$ scheduled before i in σ' must be sequenced before job B in σ'' . $\square$

Considering the two previous lemma we can derive an improved enumeration algorithm of the set of strict Pareto optima for the $Q|p_i = p|\#(L_{max}^A, C_{max}^B)$ problem. After the first iteration of Algorithm 2, we get an initial solution σ'_ε which actually is a solution of the lexicographical $Q|p_i = p|Lex(C_{max}^B, L_{max}^A)$ problem where C_{max}^B is minimized first. Then the next strict Pareto schedule σ''_ε is obtained by identifying the last operation $i \in A$ with lateness equal to $L_{max}^A(\sigma'_\varepsilon)$ and by moving all operations $j \in A$, such that $d_j \leq d_i$, to set A' in schedule σ''_ε . This way a new schedule σ''_ε with $L_{max}^A < L_{max}^A$ and $\varepsilon = C_{max}^B > C_{max}^B$ is built. The order of the operations of job A still follows the EDD order. This process is iterated until the stopping condition stated in Lemma 5 is reached (see Algorithm 3).

The complexity in the worst case of Algorithm 3 is the same than that of Algorithm 2, i.e. $\mathcal{O}((n_A + n_B) \log(m) + n_A \log(n_A))$ time, however here we perform only as many iterations as strictly non-dominated solutions exist. This means that the practical time complexity is reduced.

Now, we return back to Lemma 2 which states that there are at most $(n_A + 1)$ strictly non-dominated criteria vectors and we establish the following result.

Lemma 7 *For the $Q|p_i = p|\#(L_{max}^A, C_{max}^B)$ problem there at most $(n_A + 1)$ strictly non-dominated criteria vectors and the bound is tight.*

Proof We show that the bound is tight by means of the following example. Consider a single machine problem, i.e. $m = 1$, for which all operations of jobs A and

Algorithm 3: Solution of the $Q|p_i = p|\#(L_{max}^A, C_{max}^B)$ problem.

Data: An instance of the $Q|p_i = p|(L_{max}^A, C_{max}^B)$ problem.
Result: Strict Pareto set $\mathcal{P}$.

```

1 begin
2   Sort operations in job  $A$  in ascending order of their due dates;
3    $\mathcal{P} \leftarrow \emptyset$ ;                                     /* Initialize Pareto set. */
4    $\varepsilon \leftarrow t_{n_B}, \quad n_\varepsilon \leftarrow n_B$ ;      /* Set initial  $\varepsilon$  and  $n_\varepsilon$  values. */
5   repeat
6      $n_\varepsilon \leftarrow \operatorname{argmax}_{k=n_\varepsilon, \dots, n} \{t_k \leq \varepsilon\}$ ; /* Index of the last feasible timeslot
7     for the operation in job  $B$ . */
8      $A' \leftarrow \{\text{First } n_\varepsilon - n_B \text{ operations in } A\}$ ; /* Early operations in job  $A$ . */
9      $A'' \leftarrow A \setminus A'$ ; /* Late operations in job  $A$ . */
10    Sequence operations in job  $A'$  to the  $t_1, \dots, t_{n_\varepsilon - n_B}$  timeslots of  $\sigma_\varepsilon$ ;
11    Put the whole job  $B$  into  $t_{n_\varepsilon - n_B + 1}, \dots, t_{n_\varepsilon}$  timeslots;
12    Sequence operation in job  $A''$  to the rest  $t_{n_\varepsilon + 1}, \dots, t_n$  slots of  $\sigma_\varepsilon$ ;
13    This results to the schedule  $\sigma_\varepsilon$  with criteria  $L_{max}^A$  and  $C_{max}^B = t_{n_\varepsilon}$ ;
14     $\mathcal{P} \leftarrow \mathcal{P} \cup \{\sigma_\varepsilon\}$ ; /* Append  $\sigma_\varepsilon$  to the Pareto set. */
15     $\mathcal{I}' \leftarrow \{i \in A'' \mid C_i - d_i = L_{max}^A\}$ ; /* Determine indexes of jobs issuing
16     $L_{max}^A$  value. */
17     $\varepsilon \leftarrow \max_{i \in \mathcal{I}'} C_i$ ; /* Update  $\varepsilon$ -constraint value. */
18  until  $\mathcal{I}' \neq \emptyset$ ;
19  return  $\mathcal{P}$ ;
20 end

```

B have $p = 1$. Let us define the due dates for operations in A as follows

$$d_i = i + \frac{(i-1)n_B}{n_A}, \quad 1 \leq i \leq n_A.$$

The timeslots are defined by $t_l = l$ for all $1 \leq l \leq n$, which implies that if job B is scheduled first we get the following maximum lateness for job A :

$$L_{max}^{1A} = \max\{n_B + 1 - 1, n_B + 2 - 2 - \frac{n_B}{n_A}, \dots, n - n_A - \frac{(n_A - 1)n_B}{n_A}\} = n_B$$

with $C_{max}^{1B} = n_B$. By right timeshifting job B of one timeslot we obtain the second non-dominated solution with

$$L_{max}^{2A} = \max\{1 - 1, n_B + 2 - 2 - \frac{n_B}{n_A}, \dots, n - n_A - \frac{(n_A - 1)n_B}{n_A}\} = n_B - \frac{n_B}{n_A}$$

and $C_{max}^{2B} = n_B + 1$. By doing this repeatedly we can build the $(n_A + 1)$ non-dominated solutions with the last one having

$$L_{max}^{n_A+1A} = \max\{1 - 1, 2 - 2 - \frac{n_B}{n_A}, \dots, n_A - n_A - \frac{(n_A - 1)n_B}{n_A}\} = 0$$

and $C_{max}^{n_A+1B} = n$ criteria values. □

5 Conclusions

In this paper we have tackled the problem of scheduling two interfering job sets on uniform parallel machines. We consider the case where one job is evaluated by means of the makespan criterion whilst the other is evaluated by means of a general cost functions. An other important feature of this problem relies on the jobs which are made up of equal-size operations. To the best of our knowledge this general problem has never been dealt with in the literature. We provide polynomial time algorithms for the enumeration of the set of strict Pareto optima in the cases where the general cost function is a max-cost function or a sum-cost function. We also provide some particularizations to classic scheduling criteria as the maximum lateness or the weighted sum of completion times.

Acknowledgements The research has been partially supported by the program "Center for Mathematical and Computational Modelling (CM)²".

References

1. A. Agnetis, P.B. Mirchandani, D. Pacciarelli, and A. Pacifici. Nondominated schedules for a job-shop with two competing users. *Computational & Mathematical Organization Theory*, 6(2):191–217, 2000.
2. A. Agnetis, P.B. Mirchandani, D. Pacciarelli, and A. Pacifici. Scheduling problems with two competing agents. *Operations Research*, 42(2):229–242, 2004.
3. K. Baker and J.C. Smith. A multiple-criterion model for machine scheduling. *Journal of Scheduling*, 6:7–16, 2003.
4. P. Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *J. of Scheduling*, 2:245–252, 1999.
5. P. Baptiste. Scheduling equal-length jobs on identical parallel machines. *Discrete Appl. Math.*, 103(1-3):21–32, 2000.
6. P. Baptiste. A note on scheduling multiprocessor tasks with identical processing times. *Computers & Operations Research*, 30(13):2071–2078, 2003.
7. P. Baptiste and P. Brucker. Scheduling equal processing time jobs. In *JY Leung (Ed.): Handbook of scheduling: Algorithms, Models and Performance Analysis*, Chapman & Hall / CRC Computer and Information Science series, 2004.
8. P. Baptiste, M. Chrobak, C. Durr, and F. Sourd. Preemptive multi-machine scheduling of equal-length jobs to minimize the average flow time. *Research report, Cornell University Library* (<http://arxiv.org/abs/cs/0412094/>), (cs.DS/0605078), 2008.
9. P. Brucker and S.A. Kravchenko. Scheduling jobs with equal processing times and time windows on identical parallel machines. *J. of Scheduling*, 11:229–237, 2008.
10. T.C.E. Cheng, C.T. Ng, and J.J. Yuan. Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs. *Theoretical Computer Science*, 362:273–281, 2006.
11. M. Chrobak, C. Durr, W. Jawor, L. Kowalik, and M. Kurowski. A note on scheduling equal-length jobs to maximize throughput. *J. of Scheduling*, 9:71–73, 2006.
12. M. I. Dessouky, B. J. Lageweg, J. K. Lenstra, and S. L. van de Velde. Scheduling identical jobs on uniform parallel machines. *Statistica Neerlandica*, 44(3):115–123, 1990.
13. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.
14. E. L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546, January 1973.
15. V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms*. 2nd edition, Springer (Berlin), 2006.
16. A. Tuzikov, M. Makhaniok, and R. Männer. Bicriterion scheduling of identical processing time jobs by uniform processors. *Computers and Operations Research*, 25(1):31–35, 1998.
17. J.J. Yuan, W.P. Shang, and Q. Feng. A note on the scheduling with two families of jobs. *Journal of Scheduling*, 8:537–542, 2005.

Lagrangian Relaxation and Cut Generation for Sequence Dependent Setup Time Flowshop Scheduling Problems

Tatsushi Nishi* · Yuichiro Hiranaka

Abstract Lagrangian relaxation technique is successfully applied to solve sequence dependent setup time flowshop problem to minimize the total weighted tardiness. The relaxed problem can be decomposed into each job-level subproblem that can be effectively solved by dynamic programming. Two types of the additional constraints for the violation of sequence dependent setup time constraints are imposed to the relaxed problem to strengthen the lower bound. The decomposed subproblem with the additional constraints is also effectively solved by the novel dynamic programming. Computational results show that the lower bound derived by the proposed method is extremely better than that of branch and bound algorithm.

1 Introduction

The sequence dependent setup time flowshop scheduling problem to minimize total weighted tardiness is known to be NP-hard combinatorial optimization problem. Many industrial scheduling problems can be modeled as the sequence dependent setup time (SDST) flowshop scheduling problems. The problem treated in this study is SDST flowshop where there is a set of jobs to be processed on multi-stage flowshop where each stage is composed of single machine. A job consists of a set of operations which have to be processed sequentially for plural stages. Each operation has a fixed processing time where preemption and splitting are not allowed. The sequence of operations is the same for all stages. Set up time is incurred before the processing an operation depending on the operation processed just before. The objective function to be minimized is the total weighted tardiness.

Exact algorithms and heuristic algorithms have been studied for solving SDST flowshop problems. A well-known heuristic is NEH algorithm for m -machine, n -job

Corresponding author to be addressed

*Tatsushi Nishi

Osaka University

E-mail: nishi@sys.es.osaka-u.ac.jp

Yuichiro Hiranaka

Osaka University

E-mail: hiranaka@inulab.sys.es.osaka-u.ac.jp

flowshop proposed by Nawaz et al. (1983). The iterated greedy search algorithm based on the use of NEH heuristic by Ruiz and Stützle (2008), randomized search combined with tabu search by Eren and Grüner (2006), and genetic algorithm of Ruiz and Stützle (2006) have been reported. For exact algorithms, branch and bound with dominance elimination criterion by Ríos-Mercado and Bard (1999), a branch and cut algorithm of Stecco et al. (2008) have been reported for the minimization of makespan. These exact algorithms can solve the problems optimally only with the limited size of problems, e.g. within 10 jobs, and few machines for SDST flowshop. Heuristic algorithms can handle large-sized problems. However, the solution for those algorithms often can be trapped into bad local optimum, and heuristic algorithms cannot evaluate the optimality of solutions. Therefore it is required to derive good bounds with reasonable time.

Lagrangian relaxation (LR) is used to obtain a good lower bound that can successfully be applied to derive a near optimal solution for single machine and parallel machine scheduling problems by Luh et al. (1990), a practical jobshop scheduling problems of Hoitomt et al. (1993). In the LR, the machine capacity constraints are relaxed by Lagrange multipliers to decompose the relaxed problem into job-level subproblem that can effectively be solved by dynamic programming (Chen et al. 1995). Subgradient optimization is used to solve Lagrangian dual problem. A critical issue to be solved for standard LR technique for scheduling is slow convergence of lower bound computation due to the existence of duality gaps. To strengthen the lower bound, Lagrangian relaxation and cut generation has been developed for flowshop problems to minimize the weighted tardiness by Nishi et al. (2007). Cuts for the capacity constraint violation are generated and imposed to the relaxed problem to strengthen the lower bound. This paper concentrates on the Lagrangian relaxation combined with cut generation for SDST flowshop to minimize the total weighted tardiness. The first step of the algorithm is to decompose the original problem into individual job-level subproblem as addressed in Nishi et al. (2007). The main difference between the study and Nishi et al. (2007) is that the sequence dependent setup constraints are taken into account in this work. If the problem is decomposed into job-level subproblem, the setup constraints cannot be easily handled. To consider the setup time constraints for the SDST flowshop, new types of additional constraints are developed and imposed on the decomposed subproblem to improve the bound.

In this paper we propose Lagrangian relaxation and cut generation for SDST flowshop with total weighted tardiness (SDST-WT). The original problem is decomposed into job-level subproblems by relaxing machine capacity constraints and setup time constraints. Valid inequalities are generated to improve the lower bound. The contribution of the paper is stated as follows. We show that the SDST-WT flowshop problem is decomposable into independent job-level subproblems when capacity constraints are relaxed by Lagrange multipliers and some of setup time constraints are eliminated. Two types of cuts are created to strengthen the lower bound derived by solving relaxed problem. Computational experiments show that the proposed method can derive relatively small duality gap for SDST flowshop with up to 50 jobs and 3 machines within 2,000 seconds.

2 Problem definition and formulation

The problem treated in this study is permutation flowshop scheduling problem with N jobs and L stages where each stage has single machine to minimize the total weighted

tardiness. Sequence dependent setup time is incurred before the processing of the operation when different job is processed successively after an operation of job. The decision variables and constants for the problem are as follows.

Decision variables:

- $c_{i,l}$: completion time for the operation of job i at stage l
- $s_{i,l}$: setup time for the operation of job i at stage l
- $m_{i,l}$: machine allocated for job i at stage l
- $\delta_{j,i}$: 0-1 binary variable which takes the value 1 if the operation of job i is processed immediately after the operation of job j

Parameters:

- d_i : due date of job i
- w_i : weight of job i
- $p_{i,l}$: processing time for the operation of job i at stage l
- H : time horizon
- N : number of jobs
- L : total number of stages
- M : total number of machines
- $S_{j,i,l}$: setup time incurred for the operation of job i immediately after the operation of job j at stage l
- $\mathcal{M}_{i,l}$: set of machines which can process the operation of job i at stage l .

Let job 0 be a dummy job representing the job before the first job is processed. It is assumed that $\delta_{j,i} = 0$ if $j = i$, $c_{0,l} = s_{0,l} = m_{0,l} = \delta_{j,0} = 0$, $d_0 = w_0 = p_{0,l} = 0$. To decompose the problem into job-level subproblem, we define the decision variable for the completion time for the operation of job j at stage l : $c_{j,l}$, and the decision variable for the selection of setup time for job j : $\delta_{j,i} \in \{0, 1\}$ which takes the value 1 if the operation for job i selects the setup time before the operation of job i just after the operation of job j and zero otherwise. Let $\varphi(\tau)$ be the function where $\varphi(\tau)$ takes the value $\varphi(\tau) = 1$ if $\tau \geq 0$, and $\varphi(\tau) = 0$ otherwise, ζ_u be the pair of job i and stage l where the available machine which can process the operation of job i at stage l . These functions can be written as:

$$\varphi(\tau) = \begin{cases} 1, & \text{if } \tau \geq 0 \\ 0, & \text{otherwise,} \end{cases}$$

$$\zeta_u = \{(i, l) \mid m_{i,l} = u\}.$$

Using the functions, the SDST flowshop scheduling problem with the objective of the minimization of total weighted tardiness (P) can be formulated as the following equations.

$$(P) \quad \min \sum_{i=1}^N w_i T_i \quad (1)$$

$$\text{s. t. } T_i = \max\{0, c_{i,L} - d_i\}, \quad i = 1, \dots, N, \quad (2)$$

$$c_{i,l} \geq s_{i,l} + p_{i,l}, \quad i = 1, \dots, N, \quad l = 1, \dots, L, \quad (3)$$

$$c_{i,l-1} \leq c_{i,l} - p_{i,l}, \quad i = 1, \dots, N, \quad l = 2, \dots, L, \quad (4)$$

$$\sum_{(i,l) \in \zeta_u} \{\varphi(\tau - c_{i,l} + p_{i,l} + s_{i,l} - 1) - \varphi(\tau - c_{i,l} - 1)\} \leq 1,$$

$$\tau = 1, \dots, H, \quad u = 1, \dots, M, \quad (5)$$

$$s_{i,l} = \sum_{j=0}^N S_{j,i,l} \delta_{j,i}, \quad i = 1, \dots, N, \quad l = 1, \dots, L, \quad (6)$$

$$\sum_{j=0}^N \delta_{j,i} = 1, \quad i = 1, \dots, N, \quad (7)$$

$$\sum_{i=1}^N \delta_{j,i} \leq 1, \quad j = 0, \dots, N, \quad (8)$$

$$B(1 - \delta_{j,i}) + c_{i,l} - p_{i,l} - s_{i,l} \geq c_{j,l}, \\ j = 0, \dots, N, \quad i = 1, \dots, N, \quad l = 1, \dots, L. \quad (9)$$

B is a sufficiently large constant. (1) is the objective function of the sum of the total weighted tardiness. (2) defines tardiness penalty. (3) restricts the completion time for each job at each stage. (4) represents the technical precedence constraints for each job at each stage. (5) is the machine capacity constraints ensuring that each machine can process only one operation at the same time during the processing. (6) expresses the setup time constraints for each job at each stage. (7) ensures that each job can select only one pre-setup time. (8) represents that the number of jobs selected for the setup time just after the operation of job j is equal or less than 1. (9) specifies the condition that the completion time for each job is consistent with the selection of setup time for each job.

3 Lagrangian relaxation and dynamic programming

3.1 Lagrangian relaxation

The problem (P) is NP-hard problem in strong sense. If the constraints (5), (8) and (9) are relaxed, the original problem (P) can be decomposed into each job-level sub-problem that can effectively solved in polynomial time. The relaxed problem obtained from the Lagrangian relaxation of (5) with non-negative Lagrange multiplier $\lambda_{\tau, m_{i,l}}$, the Lagrangian relaxation of (8) with non-negative Lagrange multiplier μ_j , and the elimination of constraints (9) is formulated as follows.

$$(RP) \quad \min L(\lambda, \mu) \quad (10)$$

s. t.

$$L(\lambda, \mu) = \sum_{i=1}^N \left\{ w_i T_i + \sum_{l=1}^L \sum_{\tau=c_{i,l}-p_{i,l}-s_{i,l}+1}^{c_{i,l}} \lambda_{\tau, m_{i,l}} \right. \\ \left. + \sum_{j=0}^N \mu_j \delta_{j,i} \right\} - \sum_{u=1}^M \sum_{\tau=1}^H \lambda_{\tau, u} - \sum_{j=0}^N \mu_j \quad (11)$$

$$\lambda_{\tau, u} \geq 0, \quad \tau = 1, \dots, H, \quad u = 1, \dots, M, \quad (12)$$

$$\mu_j \geq 0, \quad j = 0, \dots, N, \quad (13)$$

(2), (3), (4), (6), (7).

The decomposed subproblem (SP_i) for each job i is described as follows.

$$(SP_i) \quad \min L_i(\lambda, \mu) \quad (14)$$

$$L_i(\lambda, \mu) = \min \left\{ w_i T_i + \sum_{l=1}^L \sum_{\tau=c_{i,l}-p_{i,l}-s_{i,l}+1}^{c_{i,l}} \lambda_{\tau, m_{i,l}} + \sum_{j=0}^N \mu_j \delta_{j,i} \right\} \quad (15)$$

$$T_i = \max\{0, c_{i,L} - d_i\}, \quad (16)$$

$$c_{i,l} \geq s_{i,l} + p_{i,l}, \quad l = 1, \dots, L, \quad (17)$$

$$s_{i,l} = \sum_{j=0}^N S_{j,i,l} \delta_{j,i}, \quad l = 1, \dots, L, \quad (18)$$

$$\sum_{j=0}^N \delta_{j,i} = 1. \quad (19)$$

3.2 Dynamic programming for subproblem with setup time selection

The subproblem (SP_i) can be solved optimally by the dynamic programming recursion extending the idea of algorithm provided by Chen et al. (1998). Let $q_{i,l}(u, t, \delta_{j,i})$ denote the criterion value for the completion of the operation of job i on machine u at time t . Let $\mathcal{F}$ be the set of time periods satisfying $\mathcal{F} = \{t \mid t = 0, \dots, H\}$.

$$q_{i,l}(u, t, \delta_{j,i}) = \begin{cases} w_i T_i + \sum_{x=t-p_{i,l}-s_{i,l}+1}^t \lambda_{x,u} + \sum_{j=0}^N \mu_j \delta_{j,i}, & \text{if } l = L, \\ \sum_{x=t-p_{i,l}-s_{i,l}+1}^t \lambda_{x,u} + \sum_{j=0}^N \mu_j \delta_{j,i}, & \text{otherwise,} \end{cases}$$

$j = 0, \dots, N, \quad l = 1, \dots, L, \quad u \in \mathcal{M}_{i,l}, \quad t \in \mathcal{F}.$

where $\mathcal{M}_{i,l}$ is the set of machines which can process the operation of job i at stage l . The Property 1 and Property 2 hold.

[Property 1.] The solution of (SP_i) with the fixed completion time as the earliest completion time $\mathcal{T}(i, l)$ for the operation of job i at stage l ($l = 1, \dots, L$) satisfying

$$\mathcal{T}(i, l) = \begin{cases} p_{i,l} + S_{0,i,l}, & \text{if } l = 1 \\ \max\{\mathcal{T}(i, l-1) + p_{i,l}, p_{i,l} + S_{0,i,l}\}, & \text{otherwise.} \end{cases}$$

is optimal when $\delta_{0,i} = 1$ is the optimal solution for SP_i .

[Proof of Property 1.] It is clear that the objective function of the total weighted tardiness is regular and $\mathcal{T}(i, l)$ is the earliest completion time for job i . Therefore Property 1 holds.

[Property 2.] The following inequality is a valid inequality for (P):

$$B(1 - \delta_{j,i}) + c_{i,l} \geq \mathcal{H}(i, j, l),$$

$i, j = 1, \dots, N, \quad i \neq j, \quad l = 1, \dots, L. \quad (20)$

where

$$\mathcal{H}(i, j, l) = \begin{cases} \mathcal{T}'(j, l) + p_{i,l} + S_{j,i,l}, & \text{if } l = 1, \\ \max\{\mathcal{H}(i, j, l-1) + p_{i,l}, \mathcal{T}'(j, l) + p_{i,l} + S_{j,i,l}\}, & \text{otherwise.} \end{cases}$$

$$\mathcal{T}'(i, l) = \begin{cases} p_{i,l} + \min_{k=0,1,\dots,N} S_{k,i,l}, & \text{if } l = 1, \\ \max\{\mathcal{T}'(i, l-1) + p_{i,l}, p_{i,l} + \min_{k=0,1,\dots,N} S_{k,i,l}\}, & \text{otherwise.} \end{cases}$$

[**Proof of Property 2.**] $\mathcal{H}(i, j, l)$ is the earliest completion time for job i when the operation for job i is processed after the operation of job i at stage l for $i, j = 1, \dots, N$. (20) is obtained by modifying $c_{j,l}$ into $\min c_{j,l}$ in (9). This reveals that (20) is the relaxation of constraint (9). (20) is a valid inequality for (P). Therefore Property 2 holds.

In order to utilize the properties for dynamic programming recursion, the function $h_{i,l}$ is defined. $h_{i,l}(u, t, \{\delta_{j,i}\})$ is the criterion value for the completion of operation at time t for job i on machine u at stage l when $\delta_{j,i}$ is given. $h_{i,l}(u, t, \{\delta_{j,i}\})$ is obtained by fixing earliest completion time for job i if $\delta_{0,i} = 1$, and it is obtained satisfying (20) when $\delta_{j,i} = 1, j \neq 0$. The dynamic programming recursion for solving subproblem (SP_i) is described as follows.

$$h_{i,l}(u, t, \{\delta_{j,i}\}) = \begin{cases} +\infty, & \text{if } (\delta_{0,i} = 1) \wedge (t \neq \mathcal{T}(i, l)), \\ +\infty, & \text{if } (j \neq 0) \wedge (\delta_{j,i} = 1) \wedge (t < \mathcal{H}(i, j, l)), \\ +\infty, & \text{if } t < H - \sum_{k=l+1}^L p_{i,k}, \\ q_{i,l}(u, t, \{\delta_{j,i}\}), & \text{otherwise,} \end{cases}$$

$$j = 0, \dots, N, l = 1, \dots, L, u \in \mathcal{M}_{i,l}, t \in \mathcal{F}. \quad (21)$$

$$f_{i,l}(u, t, \{\delta_{j,i}\}) = \begin{cases} h_{i,l}(u, t, \{\delta_{j,i}\}), & \text{if } l = 1, \\ h_{i,l}(u, t, \{\delta_{j,i}\}) + \min_{v \in \mathcal{M}_{i,l+1}} g_{i,l+1}(v, t + p_{i,l+1}, \{\delta_{j,i}\}), & \text{otherwise,} \end{cases}$$

$$j = 0, \dots, N, l = 1, \dots, L, u \in \mathcal{M}_{i,l}, t \in \mathcal{F}. \quad (22)$$

$$g_{i,l}(u, t, \{\delta_{j,i}\}) = \begin{cases} +\infty, & \text{if } t < p_{i,l}, \\ \min_{t \leq x \leq H} f_{i,l}(u, x, \{\delta_{j,i}\}), & \text{otherwise.} \end{cases} \quad (23)$$

$f_{i,l}(u, t, \delta_{j,i})$ is the optimal criterion value of the state $(u, t, \{\delta_{j,i}\})$ for the operation of job i on machine u at stage l . The optimal state $f(u^{l*}, t^{l*}, \{\delta_{j,i}^{l*}\})$ at stage l for job i is obtained by the forward recursion.

$$(u^{1*}, t^{1*}, \{\delta_{j,i}^{1*}\}) = \arg \min_{0 \leq t \leq H, u \in \mathcal{M}_{i,1}, \{\delta_{j,i} | \forall j\}} f_{i,1}(u, t, \{\delta_{j,i}\}) \quad (24)$$

$$(u^{l*}, t^{l*}, \{\delta_{j,i}^{l*}\}) = \arg \min_{t^{(l-1)*} + p_{i,l} \leq t \leq H, u \in \mathcal{M}_{i,L}, \{\delta_{j,i}\} = \{\delta_{j,i}^{(l-1)*}\}} f_{i,l}(u, t, \{\delta_{j,i}\}). \quad (25)$$

The recursion is repeated until all of stages from stage 1 to stage L for each job i . Since the number of all combinations for $\{\delta_{j,i}\}$ is N for each job j due to $\sum_{j=0}^N \delta_{j,i} = 1$ and $\delta_{j,i} \in \{0, 1\}$, the computational complexity for solving subproblem (SP_i) is $O(NMH)$.

The Lagrangian dual problem (DP) is formulated as

$$(DP) \max Q(\lambda, \mu) \quad (26)$$

$$\text{s. t. } Q(\lambda, \mu) = \sum_{i=1}^N \min L_i(\lambda, \mu) - \sum_{u=1}^M \sum_{\tau=1}^H \lambda_{\tau,u} - \sum_{j=0}^N \mu_j, \quad (27)$$

$$(2), (3), (4), (6), (7), (12), (13).$$

The subgradient optimization is used to derive near-optimal Lagrange multipliers. α is the step size parameter.

$$\lambda_{\tau,u} = \max\{0, \lambda_{\tau,u} + \alpha \frac{UB - LB}{\sum_{u=1}^M \sum_{\tau=1}^H h_{\tau,u}^2} h_{\tau,u}\}, \quad (28)$$

$$\mu_j = \max\{0, \mu_j + \alpha \frac{UB - LB}{\sum_{j=0}^N h_j'^2} h_j'\}, \quad (29)$$

where $h_{\tau,u} = \sum_{(i,l) \in \zeta_u} \{\varphi(\tau - c_{i,l}^* + p_{i,l} + s_{i,l}^* - 1) - \varphi(\tau - c_{i,l}^* - 1)\} - 1$, $h_j' = \sum_{u=1}^N \delta_{j,i}^* - 1$. $\{c_{i,l}^*\}$, $\{s_{i,l}^*\}$, $\{\delta_{j,i}^*\}$ is the solution derived by solving subproblem SP_i .

4 Cut generation

4.1 Cuts for sequence dependent setup time constraint violation

It is extremely difficult to obtain a good lower bound for solving Lagrangian dual (DP) because the consistency violation constraints (9) for the selection of setup time with the precedence relationship of completion times are eliminated for the problem (RP) as explained in section 3. Cuts are added to the (RP) to strengthen the lower bound. Let $\mathcal{N}$ be the set of jobs except dummy job 0. The cuts for the violation of constraints are as follows:

$$\begin{aligned} &\exists(i, j) \text{ such that for } i, j \in \mathcal{N} \\ &(\delta_{j,i} + \delta_{i,j} \leq 1) \wedge (\delta_{k,i} + \delta_{k,j} \leq 1, k = 0, \dots, N) \end{aligned} \quad (30)$$

$$\begin{aligned} &\exists(i, j) \text{ such that for } i, j \in \mathcal{N} \\ &(\delta_{j,i} + \delta_{i,j} \leq 1) \wedge (c_{i,1} - p_{i,1} - s_{i,1} = c_{j,1}, \text{ if } \delta_{j,i} = 1) \\ &\wedge (c_{j,1} - p_{j,1} - s_{j,1} = c_{i,1}, \text{ if } \delta_{i,j} = 1) \end{aligned} \quad (31)$$

The main idea for the cut generation is states as follows. (30) represents the additional constraint that can eliminate the inconsistency for the selection of setup time between the solution of subproblem for job i and job j . (31) is the additional constraint that can reduce the capacity constraint violation from the selection of setup time due to the characteristic of zero idle time for the first stage in the SDST flowshop problem.

More specifically, (30) ensures that the solution with $\delta_{j,i} = \delta_{i,j} = 1$ is not satisfied and the setup time cannot be selected two or more than two jobs for job k at a time, that is clearly a valid cut because all feasible solution for the original problem satisfy (30). (31) represents the constraint that the completion time for the operation of job i (j) is equal to the starting time for the operation of job j (i) at stage 1 if job i (j) selects the setup time for job j (i). For flowshop problems, idle time at stage 1 is always

zero for regular objective function. (31) restricts that the completion time for job j is equal to the starting time for job i minus setup time for job i if the setup time for the operation of job i immediately after the operation of job j is selected at stage 1, and vice versa. The lower bound can be strengthened if the cuts (30), (31) are added to (RP) because the feasible region for (RP) is reduced.

4.2 Cut generation method

Both of (30) and (31) are related to a pair of job i and job j . The cut (30) and (31) are generated at the same time for each pair of two jobs. To generate the cuts, an appropriate pair of two jobs (i, j) must be selected. In this study, the following algorithm is executed for the cut generation. The main idea is that the pair of cut is selected so that the expected difference of setup time $\Delta S_{j,i} = |S_{j,i,1} - \min_{k \in \mathcal{N} \setminus \{j\}} S_{k,i,1}|$ is maximized. This is an empirical rule derived from our preliminary experiments. The optimality of solution derived by the LR highly depends on the selection of setup time for each job. Therefore it is expected to expedite search by the selection of large expected difference of setup time.

[Cut generation algorithm]

Step 1. $\mathcal{C} = \phi$.

Step 2. Fix the solution for an optimal solution to provide the best lower bound obtained by solving each job-level subproblem.

Step 3. Enumerate a pair of jobs (i, j) ($i, j \in \mathcal{N}$) that satisfies $\delta_{j,i} = 1 \wedge c_{i,1} - p_{i,1} - s_{j,1} \neq c_{j,1}$. Let $\mathcal{E}$ denote the set of the pairs (i, j) ($i, j \in \mathcal{N}$).

Step 4. For each element in $\mathcal{E}$, calculate $\Delta S_{j,i} = |S_{j,i,1} - \min_{\{k \in (\mathcal{N} \setminus \{j\})\}} S_{k,i,1}|$. If $\delta_{i,j} = 1$, then $\Delta S_{j,i} = \Delta S_{j,i} + \Delta S_{i,j}$.

Step 5. Select a pair of (i, j) which provides the largest $\Delta S_{j,i}$ from $\mathcal{E}$. The cut for the pair of (i, j) is generated and go to Step 6. If $\mathcal{E} = \phi$, all of the pairs in $\mathcal{N} \setminus \mathcal{C}$ are set to $\mathcal{E}'$ and go to Step 7.

Step 6. $\mathcal{C} = \mathcal{C} \cup \{i, j\}$. Delete the pair of jobs which includes at least one of job i and job j from $\mathcal{E}$ and return to Step 5.

Step 7. Select a pair of (i, j) randomly from $\mathcal{E}'$. The cut for the selected pair is generated. If $\mathcal{E}' = \phi$, the algorithm is finished.

Step 8. Delete all of the pair of jobs which includes at least one of job i and job j from $\mathcal{E}'$ and return to Step 7.

Both of cuts (30) and (31) for a pair of two jobs are generated simultaneously. The number of generated cuts for (30) and (31) is restricted so that two or more than two cuts for each pair of job (i, j) is not generated to reduce the total computing time. However, it is ensured that at least a cut for (30) and (31) is generated for every job.

Example. Consider an example of SDST flowshop with 3 jobs and 2 stages. Each stage has single machine. Fig. 1(a) shows a tentative result obtained by solving (DP). Assume the solution $\delta_{2,1} = 1$, $\delta_{1,2} = 1$, $\delta_{1,3} = 1$ when the setup time for job 1 is $S_{0,1,1} = S_{0,1,2} = 5$, $S_{2,1,1} = S_{2,2,2} = 2$, $S_{3,1,1} = S_{3,1,2} = 4$, the setup time for job 2 is $S_{0,2,1} = S_{0,2,2} = 2$, $S_{1,2,1} = S_{1,2,2} = 1$, $S_{3,2,1} = S_{3,2,2} = 5$, and the setup time for job 3 is $S_{0,3,1} = S_{0,3,2} = 7$, $S_{1,3,2} = S_{1,3,2} = 3$, $S_{2,3,1} = S_{2,3,2} = 4$. The candidate of cuts is $\epsilon = \{(1, 2), (2, 1), (1, 3)\}$. For each candidate, $\Delta S_{j,i}$ is computed. $\Delta S_{2,1} = \Delta S_{2,1} + \Delta S_{1,2} = |S_{2,1,1} - S_{3,1,1}| + |S_{1,2,1} - S_{0,2,1}| = 3$, $\Delta S_{1,3} = |S_{1,3,1} - S_{2,3,1}| = 1$.

(1,2) is selected because $\Delta S_{2,1}$ is the maximum. $\mathcal{C} = \{1, 2\}$ and $\epsilon = \phi$ because the element including 1 or 2 is excluded from ϵ . $\epsilon' = \mathcal{N} \setminus \mathcal{C} = \phi$ then the algorithm is finished. An example of results with cut (1,2) is shown in Fig. 1(b).

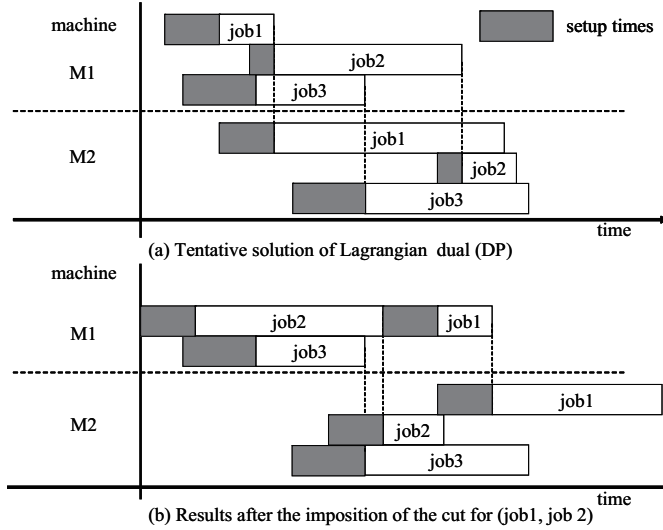


Fig. 1 Example of cut generation for SDST flowshop

4.3 Dynamic programming with cut generation

The dynamic programming recursion for the relaxed problem with the additional constraints on any two jobs is explained in this section. To solve the subproblem with cut on the operations for two jobs $i, j \in \mathcal{C}$, the recursion for the operation of job i at stage l is as follows.

$$f'_{i,l}(u, t, \{\delta_{k,i}\}) = \begin{cases} f_{i,l}(u, t, \{\delta_{k,i}\}) + \min_{t' \in \mathcal{F}, \{\delta_{k',j} \mid \forall k', k' \neq k\}} f_{j,l}(u, t', \{\delta_{k',j}\}), & \text{if } (l = 1) \wedge (\delta_{j,i} = \delta_{i,j} = 0), \\ f_{i,l}(u, t, \{\delta_{k,i}\}) + \min_{\{\delta_{k',j} \mid \forall k', k' \neq k\}} f_{j,l}(u, t - p_{i,l} - s_{i,l}, \{\delta_{k',j}\}), & \text{if } (l = 1) \wedge (\delta_{j,i} = 1) \wedge (\delta_{i,j} = 0), \\ f_{i,l}(u, t, \{\delta_{k,i}\}) + f_{j,l}(u, t + p_{j,l} + s_{j,l}, \{\delta_{k',j}\}), & \text{if } (l = 1) \wedge (\delta_{j,i} = 0) \wedge (\delta_{i,j} = 1), \\ f_{i,l}(u, t, \{\delta_{k,i}\}), & \text{if } l \neq 1 \\ +\infty, & \text{otherwise,} \end{cases}$$

$$k, k' = 0, \dots, N, l = 1, \dots, L, u \in \mathcal{M}_{i,l}, t \in \mathcal{F}. \quad (32)$$

$f'_{i,l}(u, t, \{\delta_{k,i}\})$ is the optimal criteria value for job i and job j which is the sum of the optimal cost for the state $(u, t, \{\delta_{k,i}\})$ of job i and the optimal cost for the state

$(u, t, \{\delta_{k,i}\})$ of job j under the constraint between the operation of job i and job j at stage l . There are three cases if the stage $l = 1$. The first recursion implies that the optimal cost is calculated by selecting the minimum cost if $(\delta_{j,i} = 0) \wedge (\delta_{i,j} = 0)$. The second recursion implies that job i selects the setup time that requires immediately after the processing of the operation of job j . The third recursion implies that job j selects the setup time that requires immediately after the processing of the operation of job i . The cut (30) is realized by setting $f'_{i,l}(u, t, \{\delta_{k,i}\}) = \infty$ if $(\delta_{j,i} = 1) \wedge (\delta_{i,j} = 1)$ or $(\delta_{k,i} = 1) \wedge (\delta_{k,j} = 1)$. The cut (31) is realized by the second and third recursions in (32). The optimal state with the constraints can be obtained by forward recursion by (24) and (25) in the same manner.

The optimal criteria value for the operation of job j at stage l is calculated as follows.

$$f'_{j,l}(u, t, \{\delta_{k,j}\}) = \begin{cases} f_{j,l}(u, t, \{\delta_{k,j}\}) + \min_{t' \in \mathcal{F}, \{\delta_{k',i} | \forall k', k' \neq k\}} f_{i,l}(u, t', \{\delta_{k',i}\}), & \text{if } (l = 1) \wedge (\delta_{i,j} = \delta_{j,i} = 0), \\ f_{j,l}(u, t, \{\delta_{k,j}\}) + \min_{\{\delta_{k',i} | \forall k', k' \neq k\}} f_{i,l}(u, t - p_{j,l} - s_{j,l}, \{\delta_{k',i}\}), & \text{if } (l = 1) \wedge (\delta_{i,j} = 1) \wedge (\delta_{j,i} = 0), \\ f_{j,l}(u, t, \{\delta_{k,j}\}) + f_{i,l}(u, t + p_{i,l} + s_{i,l}, \{\delta_{k',i}\}), & \text{if } (l = 1) \wedge (\delta_{i,j} = 0) \wedge (\delta_{j,i} = 1), \\ f_{j,l}(u, t, \{\delta_{k,j}\}), & \text{if } l \neq 1, \\ +\infty, & \text{otherwise,} \end{cases}$$

$$k, k' = 0, \dots, N, l = 1, \dots, L, u \in \mathcal{M}_{j,l}, t \in \mathcal{F}. \quad (33)$$

The recursion for the operation of job j at stage l can be computed in the same way as (32). The computing time for solving subproblem for job i and job j with additional constraints is $O(N^2MH)$. It is possible to construct the same type of cuts for any n jobs. The computing time is $O(N^nMH)$. The cuts up to any 3 jobs, any 4 jobs are applied for our numerical experiments.

5 Overall algorithm of Lagrangian relaxation and cut generation

The proposed algorithm is explained in this section. To explain the algorithm, the ordinary Lagrangian relaxation (LR) based on job-based decomposition is explained as follows.

[LR algorithm]

Step 1. Set the Lagrange multipliers (e.g. $\lambda_{\tau,u} = 0, \mu_j = 0$).

Step 2. Solve the subproblem (SP_i) by the dynamic programming recursion of (22) and compute a lower bound LB .

Step 3. Construct a feasible solution by NEH algorithm combined with local search and compute an upper bound UB .

Step 4. If $GAP = UB - LB$ is less than 1 or the lower bound is not updated for a fixed time, the algorithm is finished.

Step 5. Update the Lagrange multipliers by (28) and (29). Then return to Step 2.

The proposed algorithm consists of the following steps. There are two types of Lagrange multipliers μ_j and $\lambda_{\tau,u}$. To solve the Lagrangian dual problem efficiently,

Lagrange multiplier is independently optimized. The overall algorithm of the proposed method consists of the following steps.

[Proposed algorithm]

- Step 1.** Construction of a feasible solution by **[NEH algorithm combined with local search]**. Set an initial Lagrange multipliers.
- Step 2.** Execute **[LR algorithm]** with a fixed μ_j as the solution with the best lower bound. Only $\lambda_{\tau,u}$ is optimized by the subgradient method. The dynamic programming recursions of (32), (33) are used to solve the subproblem with cuts in the LR algorithm.
- Step 3.** Execute **[LR algorithm]** with fixed $\lambda_{\tau,u}$ as the solution with the best lower bound. Only μ_j is optimized by the subgradient method. The dynamic programming recursions of (32), (33) are used to solve the subproblem with cuts in the LR algorithm.
- Step 4.** If $GAP = UB - LB$ is less than 1, or the lower bound has not been updated for predetermined number of times, the algorithm is finished. Otherwise return to Step 2.
- Step 5.** Set the Lagrange multipliers with the best lower bound.
- Step 6.** Execute **[Cut generation algorithm]**.
- Step 7.** Execute **[LR algorithm]** with fixed μ_j as the solution with the best lower bound. Only $\lambda_{\tau,u}$ is optimized by the subgradient method. The dynamic programming recursions of (32), (33) are used to solve the subproblem with cuts in the LR algorithm.
- Step 8.** Execute **[LR algorithm]** with fixed $\lambda_{\tau,u}$ as the solution with the best lower bound. Only μ_j is optimized by the subgradient method. The dynamic programming recursions of (32), (33) are used to solve the subproblem with cuts in the LR algorithm.
- Step 9.** If $GAP = UB - LB$ is less than 1, or the lower bound has not been updated for predetermined number of times, the algorithm is finished. Otherwise return to Step 7.

In the proposed algorithm, NEH algorithm combined with local search[1] is used to derive an upper bound only in the first step. The algorithm to derive an upper bound consists of the following steps.

[NEH algorithm combined with local search]

- Step 1.** Let $\mathcal{S}'$ be the sequence of jobs are sorted according to the earliest starting time in ascending order.
- Step 2.** $k := 1$, $\mathcal{S} = \mathcal{S}'$, $UB = +\infty$.
- Step 3.** Select 5 jobs from $\mathcal{S}$. $\mathcal{R}$ is the set of the selected 5 jobs.
- Step 4.** $k' := 1$, $r' := 1$, $UB' := +\infty$.
- Step 5.** Select a job j randomly from $\mathcal{R}$. The selected job is inserted into the k' th position in $\mathcal{S}'$. A feasible schedule is obtained by dispatching the set of jobs $\mathcal{S}'$ in forward. Based on the schedule, an upper bound UB'' is calculated.
- Step 6.** If $UB'' \leq UB'$, then $UB' = UB''$ and $r' = k'$. Delete job j from $\mathcal{S}'$, $k' := k' + 1$. If $k' \leq |\mathcal{S}'| + 1$ then return to Step 5.
- Step 7.** If $k' > |\mathcal{S}'| + 1$, then job j is inserted in the r' th position in $\mathcal{S}'$. $\mathcal{R} = \mathcal{R} \setminus j$ and return to Step 4. If $\mathcal{R} = \emptyset$ and $UB' \leq UB$, then $UB = UB'$, $\mathcal{S} = \mathcal{S}'$, $k := k + 1$. If k is more than predetermined value, the algorithm is finished. Otherwise return to Step 3.

6 Computational experiments

Computational experiments are executed for the SDST flowshop with total weighted tardiness (SDST-WT) and the SDST flowshop with total weighted flowtime (SDST-WFT) problems. The flowshop with total weighted flowtime can be realized easily by setting the due date for all jobs zero. The parameters for the problem instances are shown in Table 1. The number of jobs N is selected from $N = \{5, 10, 20, 30, 40, 50\}$, the number of stages $L = 3$. Ten instance problems are created for each case. Setup time $S_{j,i,l}$ for each instance is selected from uniform distribution on $[1, 9]$ on the condition that $S_{j,i,1} = S_{j,i,2} = \dots = S_{j,i,L}$. The processing time is generated from uniform distribution on $[1, 20]$. The convergence condition for **[LR algorithm]** is that LB and UB is less than 1.00 or the lower bound has not been updated 50 times, and the convergence condition for **[Proposed algorithm]** at Step 4 and Step 9 is that the difference between LB and UB is less than 1.00 or the lower bound has not been updated 2 times. The condition The Intel Pentium D 3.4GHz with 1GB memory is used for computation.

The effects of cut generation are investigated for SDST-WT problems. The lower bound derived by the proposed method with no cuts (LB with no cut), the proposed method with cut (30) (LB with cut1), the proposed method with cut (31) (LB with cut2), the proposed method with cut (30) and (31) (LB with cut1+cut2) are summarized in Table 2. The results of computation time are summarized in Table 3. The results demonstrate that LB with cut2 is more better than LB with cut1 for all problems and cut1+cut2 is more effective because the total CPU time for cut1+cut2 is almost the same as that of cut2. The performance of cut1 is not effective for large-sized problems. This is because cut1 is very weak for large-sized problems when the feasible region for relaxed problem is too large. From these results, it is demonstrated that LR with cut1+cut2 is more effective than LR with no cuts without significantly increasing the total computation time.

The performance of duality gap for the proposed method with two types of cuts is demonstrated for SDST-WT and SDST-WFT problems. Average computational results of ten cases are summarized in Tables 4, 5. The performance of average duality gap for the proposed method for SDST-WT, SDST-WFT is within 14% for all problems. Especially for SDST-WFT, duality gap is within 12% in worst cases although the total computation time is within 2000 second for all cases. The results demonstrate that the proposed method can derive near-optimal solution for SDST-WT, SDST-WFT problems.

In order to compare the performance between the other methodologies, the performance of lower bound for three types of methods are compared for the proposed method (LRCUT), branch and bound for solving integer programming problem by CPLEX10.1 (CPLEX) with maximum 3,600 seconds of computation time, and branch and bound method (B&B) with maximum 3,600 seconds of computation time. For B&B, the branching is executed by enumerating the sequence of operations and calculating the starting time of operations in forward. The lower bound for each node is computed by

$$LB_{B\&B} = \sum_{i \in N_{fix}} w_i \max\{0, c_{i,L}^{fix} - d_i\} + \sum_{i \in N_{free}} w_i \max\{0, c_{i,L}^{free} - d_i\} \quad (34)$$

where N_{fix} is the fixed job sequence, and N_{free} is the unfixed job sequence in the branching tree. $c_{i,L}^{fix}$ is the earliest completion time at the last stage L considering

Table 1 Parameters for the instance problems

Number of jobs: N	5, 10, 20, 30, 40, 50
Number of stages: L	3
Due date: d_i for SDST-WT	$[0, 2N]$
Due date: d_i for SDST-WFT	0
Weight: w_i	$[1, 10]$
Processing time: $p_{i,l}$	$[1, 20]$
Setup time: $S_{j,i,l}$	$[1, 9]$

Table 2 Effects of cut generation to the performance of lower bound

N	LB without cut	LB with cut1	LB with cut2	LB with cut1+cut2
10	3857.8	3870.4	3901.4	3901.9
20	9060.4	9067.2	9193.0	9193.6
30	21280.1	21285.3	21387.5	21385.8
40	31138	31138	31174.2	31173.7
50	45984	45984	46015.5	46015.5

Table 3 Effects of cut generation to the total computation time [sec.]

N	Time without cut	Time with cut1	Time with cut2	Time with cut1+cut2
10	9.95	12.54	12.69	12.60
20	88.49	100.3	109.2	109.2
30	296.21	327.69	338.9	339.1
40	633.14	633.14	648.1	644.3
50	1164.24	1286.47	1301.4	1301.5

precedence constraints and setup time in the fixed sequence N_{fix} , $c_{i,L}^{free}$ is the earliest completion time at the last stage L for the job in the last sequence in N_{free} . If the computation time for CPLEX and B&B is larger than 3600 sec., the NEH algorithm combined with local search is executed to derive a feasible solution.

If the number of jobs N is 5 or 10, the lower bound for CPLEX or B&B is better than that of the proposed method. However, if the number of jobs is increased, the lower bound derived by CPLEX or B&B is extremely smaller than that of the proposed method. This is because computational complexity is significantly increased if the number of jobs is increased. It is extremely difficult for CPLEX or B&B to obtain a good lower bound in realistic computation time. The proposed method can generate good lower bound even for 50 job problem in reasonable computation time. It is demonstrated that the lower bound of the proposed method is better than that of the conventional methods.

7 Concluding remarks

In this paper, we have proposed a Lagrangian relaxation and cut generation for sequence dependent setup time flowshop problem with the total weighted tardiness. The original problem has been decomposed into job-level subproblems by the Lagrangian relaxation of capacity constraints and the removal of sequence dependent setup time constraints. The additional constraints on the relaxed problem have been imposed to strengthen the lower bound. Computational experiments have demonstrated the effectiveness of the lower bound of the proposed method compared with that of the CPLEX

Table 4 Computational results for SDST-WT

N		UB	LB	Time(s)	DGAP(%)	$\frac{UB_{LRCUT}}{UB_{CPLEX}} [\%]$	$\frac{UB_{LRCUT}}{UB_{CPLEX}} [\%]$
5	LRCUT	1279.6	1256.2	2.1	2.09	98.17	98.17
5	CPLEX	1279.6	1279.6	0.33	0		
5	B&B	1279.6	1279.6	0.17	0		
10	LRCUT	4243.6	3930.1	18.68	7.87	93.92	92.61
10	CPLEX	4243.6	4184.7	1111.4	1.11		
10	B&B	4243.6	4243.6	7.11	0		
20	LRCUT	10491.3	9244.3	154.33	13.49	287.08	136.42
20	CPLEX	-	3220.1	3600	-		
20	B&B	10477.7	6776.5	3600	54.53		
30	LRCUT	24251.2	21456.1	480.9	13.1	575.49	256.77
30	CPLEX	-	3728.3	3600	-		
30	B&B	24200.6	8356.2	3600	190.32		
40	LRCUT	35196.1	31203.3	908.02	12.97	878.67	486.30
40	CPLEX	-	3551.2	3600	-		
40	B&B	35202.8	6416.5	3600	472.58		
50	LRCUT	51984.6	46016.7	1782.97	13.19	1530.27	1048.03
50	CPLEX	-	3007.1	3600	-		
50	B&B	51913.4	4390.8	3600	1138.81		

Table 5 Computational results for SDST-WFT

N		UB	LB	Time(s)	DGAP(%)	$\frac{UB_{LRCUT}}{UB_{CPLEX}} [\%]$	$\frac{UB_{LRCUT}}{UB_{CPLEX}} [\%]$
5	LRCUT	1583.1	1553.4	1.41	1.9	98.12	98.12
5	CPLEX	1583.1	1583.1	0.34	0		
5	B&B	1583.1	1583.1	0.18	0		
10	LRCUT	4719.5	4429.1	19.38	6.43	95.23	93.85
10	CPLEX	4727.9	4651.1	1149.1	1.25		
10	B&B	4719.5	4719.5	7.19	0		
20	LRCUT	13403.9	12109.3	163.67	10.68	212.35	129.67
20	CPLEX	-	5702.6	3600	-		
20	B&B	13439.6	9338.7	3600	43.2		
30	LRCUT	26641.8	23951.6	435.89	11.37	314.67	189.06
30	CPLEX	-	7611.6	3600	-		
30	B&B	26445.5	12668.8	3600	108.59		
40	LRCUT	44060.8	39501.3	984.01	11.73	400.73	269.76
40	CPLEX	-	9857.4	3600	-		
40	B&B	44120.2	14642.9	3600	202.13		
50	LRCUT	69755.6	63190.2	1803.24	10.48	510.44	393.88
50	CPLEX	-	12379.6	3600	-		
50	B&B	68941.6	16042.9	3600	330.41		

and the standard branch and bound algorithm. Future work is to eliminate duality gap for large-sized problems by the consideration of cuts with reasonable computation time.

References

1. R. Ruiz, T. Stützle, "An iterated greedy heuristic for the sequence dependent setup times flowshop scheduling problem with makespan and weighted tardiness objectives", *European Journal of Operational Research*, vol. 187, pp. 1143–1159, 2008.
2. T. Eren, E. Güner, "A bicriteria flowshop scheduling problem with setup times", *Applied Mathematics and Computation*, vol. 183, pp. 1292–1300, 2006.
3. R. Ruiz, C. Maroto, J. Alcaraz, "Solving the flowshop scheduling problem with sequence dependent setup times using advances metaheuristics", *European Journal of Operational Research*, vol. 165, pp. 34–54, 2005.

4. M. Nawaz, E.E. Ensore Jr., I. Ham, "A heuristic algorithm for the m-Machine n-Job Flow-shop Sequencing Problem", *OMEGA*, vol. 11, no. 1, pp. 91-95, 1983.
5. X. Luo, F. Chu, "A branch and bound algorithm of the single machine schedule with sequence dependent setup times for minimizing total tardiness", *Applied Mathematics and Computation*, vol. 183, pp. 575-588, 2006.
6. P.L. Rocha, M.G. Ravetti, G.R. Mateus, P.M. Pardalos, "Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times", *Computers and Operations Research*, vol. 35, pp. 1250-1264, 2008.
7. T. Ibaraki and Y. Nakamura, "A dynamic programming method for single machine scheduling", *European Journal of Operational Research*, vol. 38, no. 7, pp. 1066-1079, 1993.
8. J. Hoitomt, P. B. Luh, and K. R. Pattipati, "A practical approach to job-shop scheduling problems," *IEEE Transactions on Robotics and Automation*, vol. 9, pp. 1-13, 1993.
9. P.B. Luh, D.J. Hoitomt, E. Max, and K.R. Pattipati, "Scheduling generation and reconfiguration for parallel machines," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 6, pp. 687-696, 1990.
10. T. Nishi, Y. Hiranaka, M. Inuiguchi, "A successive Lagrangian relaxation method for solving flowshop scheduling problems with total weighted tardiness," in *Proc. 3rd Annual Conference on Automation Science and Engineering*, pp. 875-880, 2007.
11. H. Chen, C. Chu and J.M. Proth, "A more efficient Lagrangian relaxation approach to job-shop scheduling problems," *IEEE Int. Conf. Robotics and Automation*, pp. 495-501, 1995.
12. H. Chen, C. Chu and J.M. Proth, "An improvement of the Lagrangian relaxation approach for job shop scheduling: A dynamic programming method," *IEEE Trans. Robot. Automat.*, vol. 13, no. 5, pp. 786-795, 1998.
13. G. Stecco, J.F. Cordeau, E. Moretti, "A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times," *Computers and Operations Research*, vol. 35, pp. 2635-2655, 2008.
14. R.Z. Ríos-Mercado, J.F. Bard, "A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times," *IIE Transactions*, vol. 31, pp. 721-731, 1999.

Abstracts

Estimation of absolute error for the resources-constrained project scheduling problem

Alexander A. Lazarev · Evgeny R. Gafarov

Abstract We consider some special cases of the *NP*-hard resource-constrained project scheduling problem (*RCPSP*) to minimize the makespan. We conjecture that the ratio of the optimal makespan of *RCPSP* to that of the preemptive version of the problem is less than 2. We show that a well-known lower bounds for the problem may yield bad approximation ratios or its calculation is an *NP*-hard problem too. We also provide some new estimates of the optimal makespan of *RCPSP*.

1 Introduction

The resource-constrained project scheduling problem (*RCPSP*) is formulated as follows. A set $N = \{1, \dots, n\}$ of n jobs is given. A constant amount of Q_k units of resource $k, k = 1, \dots, K$, is available at any time. Job $j \in N$ has to be processed for p_j time units. During this period a constant amount of q_{jk} units of resource k is occupied. Furthermore, a set of finish-start precedence relations $i \rightarrow j$ is defined between jobs i and j ($i, j \in N$) according to an acyclic directed graph G . The objective is determine the starting time S_j for each job $j = 1, \dots, n$, in such a way that:

- i at each time t the total resource required does not exceed the resource availability for each resource type;
- ii the given job precedence constraints are fulfilled;
- iii the makespan $C_{\max} = \max_{j=1}^n C_j$, where $C_j = S_j + p_j$ is the completion time of job j , is minimized.

Without lost of generality consider an augmented *RCPSP* with two "dummy" jobs 0 and $n+1$, where $p_0 = p_{n+1} = 0$ and there are precedence constraints between triad

Alexander A. Lazarev
Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya st. 65, 117997, Moscow, Russia. The work is part of the project supported by the program of Presidium of the Russian Academy of Sciences N 29 "The mathematical theory of control".
E-mail: jobmath@mail.ru

Evgeny R. Gafarov
Computing Centre of the Russian Academy of Sciences
E-mail: axel73@mail.ru

jobs $0 \rightarrow j \rightarrow n+1$, $j = 1, \dots, n$, $q_{0k} = q_{n+1k} = 0$ for each resource $k = 1, \dots, K$. Let UB be an upper bound on $C_{\max}$, e.g., $UB = \sum_{i=1}^n p_i$. Define the "time window" $[r_i, d_i]$ for each $i \in N$ as follows:

$$r_0 = 0, \quad r_i = \max_{j|(j \rightarrow i) \in G} \{r_j + p_j\}, \quad i = 1, \dots, n+1;$$

$$d_{n+1} = UB, \quad d_i = \min_{j|(i \rightarrow j) \in G} \{d_j - p_j\}, \quad i = n, n-1, \dots, 0.$$

2 Lower Bounds

Define $C_{\max}^*$ (or $C_{\max}(S^*)$, where S^* is an optimal schedule) as the optimal value for the problem without preemption and $C_{\max}^*(pmtn.)$ as the optimal value when preemption is allowed. We show that a well-known lower bound for the problem may yield bad approximation ratios or its calculation is an *NP*-hard problem [3]. Denote $LB_0 = CP$, where CP is the length of the critical path of the precedence graph G (if G is empty then $CP = p_{max}$). Also through CP we will denote the set of jobs which are include in the critical path too. It is evident that LB_0 is a lower bound on $C_{\max}^*$.

Lemma 1 *There exists an instance of RCPSP for which $\frac{C_{\max}^*}{LB_0} = n$.*

Proof. Consider an instance of *RCPSP* with a single resource and $p_i = p, i = 1, \dots, n$. There are no precedence relations (i.e., graph G is empty), and $q_{i1} = Q_1$ for each $i = 1, \dots, n$. $\square$

Obviously, $LB_0 = CP = p$. But $C_{\max}^* = pn$ according to the resource constraint. Therefore $\frac{C_{\max}^*}{LB_0} = n$.

$$\text{Let } LB_1 = \max_{k=1}^K \left[\sum_{j=1}^n q_{jk} p_j / Q_k \right].$$

Lemma 2 *There exists an instance of RCPSP for which $C_{\max}^* - LB_1 = \sum_{j=1}^n p_j - 1$.*

Proof. Consider an instance of *RCPSP* where $i \rightarrow i+1$, $i = 1, \dots, n-1$, and $q_{i1} = \varepsilon$, $i = 1, \dots, n$. Let $Q_1 = \sum_{i=1}^n p_i \varepsilon$. Obviously,

$$LB_1 = \sum_{i=1}^n q_{i1} p_i / Q_1 = 1. \text{ But } C_{\max}^* = \sum_{i=1}^n p_i. \text{ So } C_{\max}^* - LB_1 = \sum_{i=1}^n p_i - 1. \square$$

Let $LB_S = CP + \max_{j \notin \{CP\}} \{\max\{p_j - e_j, 0\}\}$, where e_j is the maximum length of an interval contained in $[r_j, d_j]$, in which j can be processed with the activities from CP simultaneously without violating the resource constraints.

Lemma 3 *There exists an instance of RCPSP for which $\frac{C_{\max}^*}{LB_S} = \frac{n}{2}$.*

Proof. Consider an instance of *RCPSP* with a single resource and $p_i = p, i = 1, \dots, n$. There are no precedence relations (i.e., graph G is empty). Let $q_{i1} = Q_1$, for each $i = 1, \dots, n$, then $LB_0 = p$. So we have lower bound $LB_S = p + \max\{p-0, 0\} = 2p$.

But $C_{\max}^* = pn$ according to the resource constraint. Therefore $\frac{C_{\max}^*}{LB_S} = \frac{n}{2}. \square$

2.1 Lower Bound of Mingozi et al.

We show that the calculation of the lower bound of Mingozi et al. [3], LB_M , is an *NP*-hard problem. We consider a linear programming formulation that partially relaxes the precedence constraints and allows preemption. The columns of this *LP* correspond to the so-called non-dominated feasible subsets. Feasible set X is set of jobs that may be presented simultaneously, i.e., there are no precedence relations between any pair of $i, j \in X$ and all the resource constraints are satisfied (i.e., $\sum_{i \in X} q_{ik} \leq Q_k$ for $k = 1, \dots, K$). Such set is called non-dominated if it is not a proper subset $X \subset Y$ of another feasible set Y . We consider all the non-dominated feasible sets and additionally the one-element sets $\{i\}$ for all $i = 1, \dots, n$.

We denote all these sets by $X_1, X_2, \dots, X_f$ (f – number of sets) and associate each set X_j with an incidence vector $a^j \in \{0, 1\}^n$ defined by $a_i^j = 1$ if $i \in X_j$, and $a_i^j = 0$ otherwise, $j = 1, \dots, f$.

Furthermore, let x_j be a variable denoting the number of time units over which all the jobs in X_j are processed simultaneously. Then the following linear program problem provides a lower bound LB_M for *RCPSP* by relaxing the precedence constraints and allowing preemption:

$$\sum_{j=1}^f x_j \longrightarrow \min \quad (1)$$

s.t.

$$\begin{cases} \sum_{j=1}^f a_i^j x_j \geq p_i, & i = 1, \dots, n; \\ x_j \geq 0, & j = 1, \dots, f. \end{cases} \quad (2)$$

Lemma 4 *The calculation of LB_M is NP-hard problem.*

Proof. Consider the well-known *NP*-hard "Packing in Pallets" problem. We have n items with lengths w_i , $i = 1, \dots, n$, and some pallets with length W . The objective is to pack the items in the pallets in such a way that the number of pallets used is minimized.

We consider a special case of *RCPSP* and its reduction to the "Packing in Pallets" problem. Each job $i = 1, \dots, n$ corresponds to an item i . Let $p_i = 1$, $q_{i1} = w_i$ for each job $i = 1, \dots, n$ and $Q_1 = W$. The graph G is empty. Then $C_{\max}(S^*)$ is equal to the minimum number of pallets.

Obviously, $LB_M = C_{\max}^*$ for the special case of *RCPSP*. So the calculation of LB_M is an *NP*-hard problem. $\square$

Lemma 5 *There exists an instance of *RCPSP* for which $\frac{C_{\max}^*}{LB_M} \approx 2$.*

Proof. Consider an instance of *RCPSP* as shown in Figure 1. For the instance, we have the following sets:

$$\begin{aligned} X_1 &= \{1, n\}, \\ X_2 &= \{2\}, \\ X_3 &= \{3, n\}, \\ X_4 &= \{4\}, \end{aligned}$$

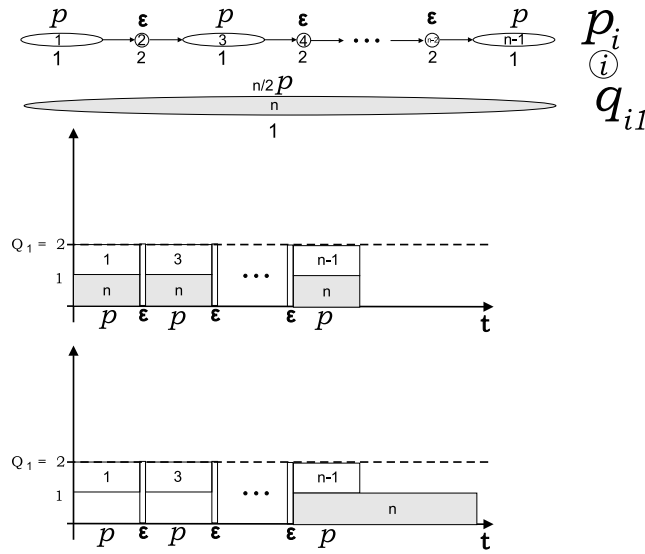


Fig. 1 Two schedules: one with preemption and other without preemption

$$\dots, \\ X_{n-1} = \{n-1, n\}$$

and the optimal solution for the corresponding *LP* is $x_1 = x_3 = \dots = x_{n-1} = p$, $x_2 = x_4 = \dots = x_{n-2} = \varepsilon$ [3].

We can vary the parameters n , p , ε for $\frac{C_{\max}^*}{LB_M} \approx 2$. $\square$

Lemma 6 *There exists an instance of RCPSP for which $\frac{C_{\max}^*}{LB_M} = 1.5$.*

Proof. Consider an instance of *RCPSP* as shown in Figure 2, where $\frac{Q_1}{2} \gg \varepsilon$. For the instance, we have the following sets: $X_1 = \{1, 4\}$, $X_2 = \{2, 3\}$, where $x_1 = p$, $x_2 = p$, and $LB_M = 2p$. But $C_{\max}(S^*) = 3p$. Therefore $\frac{C_{\max}^*}{LB_M} = 1.5$. $\square$

So we have shown that a well-known lower bound for the problem may yield bad approximation ratios or its calculation is an *NP*-hard problem.

2.2 Conjecture $C_{\max}^* < 2 \cdot C_{\max}^*(pmtn.)$

An instance of *RCPSP* is shown in Figure 1. There are a single resource with $Q_1 = 2$ and two optimal schedules with and without preemption.

The solution of *RCPSP* with preemption may be considered as a list of sets of intervals

$$\begin{aligned} S' = \{ & S_1 = \{[s_{11}, c_{11}), \dots, [s_{1m_1}, c_{1m_1})\}, \\ & S_2 = \{[s_{21}, c_{21}), \dots, [s_{2m_2}, c_{2m_2})\}, \\ & \dots \\ & S_n = \{[s_{n1}, c_{n1}), \dots, [s_{nm_n}, c_{nm_n})\} \}. \end{aligned}$$

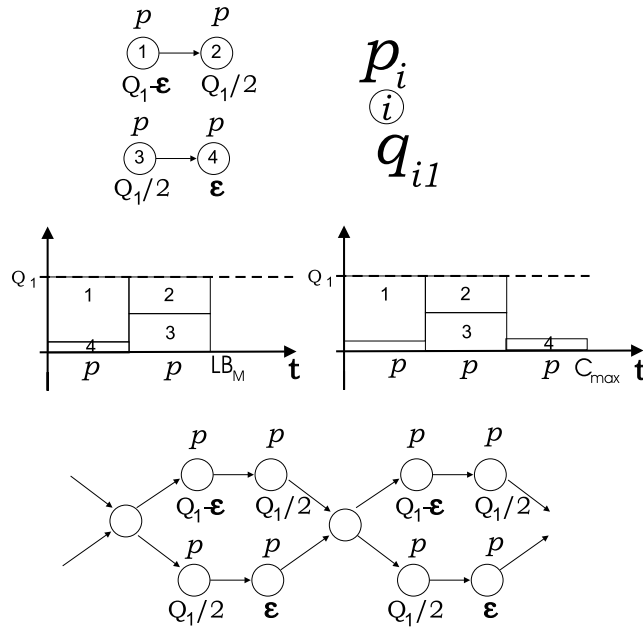


Fig. 2 $\frac{C_{\max}^*}{LB_M} = 1.5$

Job 1 is processed without preemption in the intervals $[s_{11}, c_{11}), \dots, [s_{1m_1}, c_{1m_1})$, with $m_1 - 1$ preemptions.

For *RCPSP*, we propose the following conjecture:

Conjecture 1 $C_{\max}^* < 2 \cdot C_{\max}^*(pmtn.)$

To establish the conjecture, we consider an optimal schedule S' for an instance of *RCPSP* when preemption is allowed. We then transform schedule S' into a feasible schedule S without preemption.

If we show that $C_{\max}(S) \leq 2 \cdot C_{\max}(S')$, then we have $C_{\max}(S^*) \leq C_{\max}(S) \leq 2 \cdot C_{\max}(S')$ and the conjecture is true.

If there is only one preempted job i in the schedule S' , then we can transform the schedule as follows: we assign the processing of job i to the interval $[s_{im_i}, s_{im_i} + p_i)$ without preemption. And the jobs j for which $s_{j1} > c_{im_i}$ are assigned to the intervals

$$[s_{j1} + p_i - (c_{im_i} - s_{im_i}), c_{j1} + p_i - (c_{im_i} - s_{im_i}))$$

(see Figure 3). The given job precedence constraints are fulfilled. If $j_1 \rightarrow j_2$ and $c_{j11} < s_{j21}$ (S' is a feasible schedule) then in the transformation schedule

$$c_{j11} + p_i - (c_{im_i} - s_{im_i}) < s_{j21} + p_i - (c_{im_i} - s_{im_i}).$$

According to this transformation, $C_{\max}$ increases up to $p_i - (c_{im_i} - s_{im_i})$, $j = 1, \dots, n$, i.e., $C_{\max}(S) \leq 2 \cdot C_{\max}(S')$.

In an analogous manner, we can present a transformation for the case where two jobs are preempted.

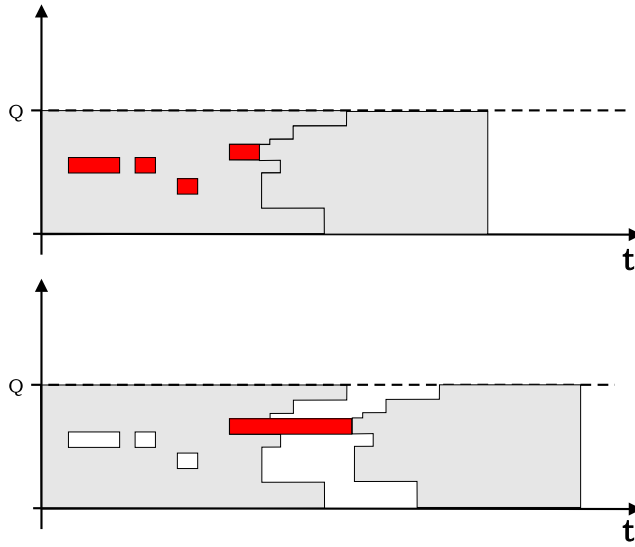


Fig. 3 Conjecture $C_{\max}^* < 2 \cdot C_{\max}^*(pmtn.)$

For the case with more than two preempted jobs, we have not been able to give a proof yet.

It is well known that the conjecture is true for a special case of the problem $Pm|prec|C_{\max}$ [4].

Consider the special case A of $RCPSP$ with a single resource (Q_1 units) and an empty graph G , i.e., the number of precedence constraints $v' = 0$. For this case, we have the following relation: $C_{\max}^* - p_{\max} < 2 \cdot C_{\max}^*(pmtn.)$.

2.3 Flat graphs of precedence relations

For several resource-constrained project scheduling problems with precedence relations, we have the following theorem.

Theorem 1 *For any instance of $RCPSP$ with n jobs and v relations, there exists an analogous instance with a flat graph G' with n' jobs and v' relations, where $n + v \geq n' + v'$.*

We obtain an analogous instance from the original instance by adding "dummy" jobs and deleting all the unnecessary relations. The proof of the theorem follows from Lemmas 1 and 2.

Lemma 7 *If there is a subgraph $G' \subset G$ that is isomorphic to the special graph $K_{3,3}$, then we can transform it into a flat subgraph by adding "dummy" jobs (with $p_j = 0$) and deleting all the unnecessary relations.*

Proof. See the transformation rules shown in Figures 4, 5 and 6. $\square$

Lemma 8 *If there is a subgraph $G' \subset G$ that is isomorphic to the special graph $K_{5,2}$, then we can transform it into a flat subgraph by deleting all the unnecessary relations.*

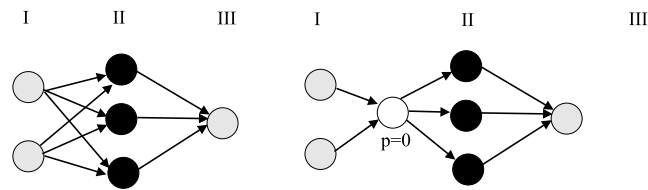


Fig. 4 Transformation of $K_{3,3}$

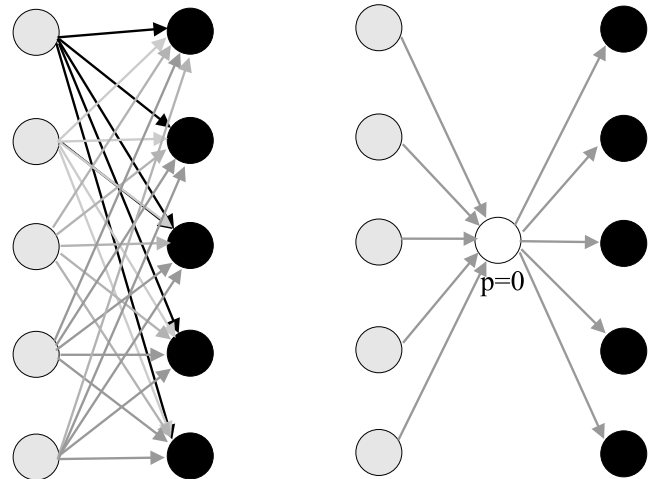


Fig. 5 Transformation of $K_{k,k}$

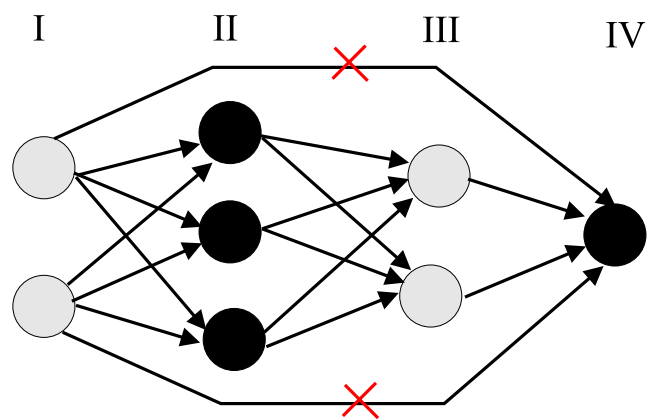


Fig. 6 Transformation of $K_{4,4}$

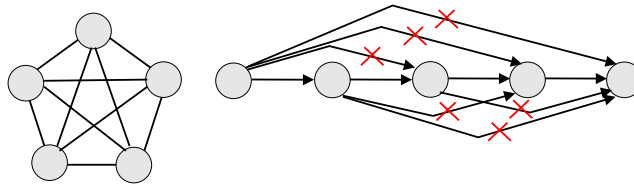


Fig. 7 Transformation of $K_{5,2}$

Proof. See the transformation rule shown in Figure 7. $\square$

This fact allows us to better express the precedence graph associated with any instance of *RCPSP* in some commercial software program products, e.g., *MSPProject*, *Primavera*, etc.

3 Conclusion

We showed that a well-known lower bound for *RCPSP* may yield bad approximation ratios or its calculation is *NP*-hard problem. We proposed the following conjecture that $C_{\max}^* < 2 \cdot C_{\max}^*(pmtn.)$. The proof (or refutation) the given hypothesis is important:

if the hypothesis is true, the received estimation (with preemption) can be used in algorithms of branch and bounds for the solution of an initial problem (without preemption);

if the hypothesis is not true and it will turn out $C_{\max}(S^*) \approx O(n) \cdot C_{\max}(S')$ (i.e. the relation of criterion functions of equally order n) "the best known" the lower estimation LB_M , used in a method of branch & bounds (branch & cuts, Constraint Programming, etc.) can take the leave from $C_{\max}(S^*)$ in $O(n)$ time. So we'll have the lower estimation LB_M will appear inefficient generally...

We showed that $2 \cdot C_{\max}^*(pmtn.) > C_{\max}^* - p_{\max}$ for a special case of *RCPSP*. For any instance of *RCPSP*, we established the existence of an analogous instance with a flat graph.

Acknowledgements The work is part of the project supported by the program of Presidium of the Russian Academy of Sciences N 29 "The mathematical theory of control".

References

1. Brucker P., Knust S. Complex Scheduling. Springer:Berlin; 2006.
2. Merkel D., Middendorf M., Schneck H. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*; 6; 333–346, 2002
3. Mingozzi A., Maniezzo V., Ricciardelli S., Bianco L. An exact algorithm for project scheduling with resource constraints based on new mathematical formulation. *Management Science*; 44; 714–729, 1998
4. Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Schmoys D.B. Sequencing and Scheduling: Algorithms and Complexity. Report BS-R8909, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands.

Polynomial complexity of the cyclic identical coupled task problem

Vassilissa Lehoux-Lebacque · Nadia Brauner ·
Gerd Finke

1 Introduction

Coupled-tasks were introduced by Shapiro in 1980 for scheduling the operations of a radar [19]. The radar emits a pulse that is transmitted to a target and reflected back to the radar, which receives the pulse. Hence the radar must process two operations per task (emission and reception), and those operations are separated by a fixed duration.

The coupled task scheduling problem is then to process tasks on a single machine (the radar) with each task j composed of two operations of lengths a_j and b_j separated by exactly L_j time units. The objective is to minimize the makespan in the non-cyclic case (given a fixed number of coupled tasks) or the throughput rate in the cyclic case (infinite number of tasks).

Coupled-task problems belong to the wider class of scheduling multi-operation tasks, where consecutive operations are separated by a certain time interval. In manufacturing processes, the time that has to elapse between operations (delays, time-lags) are often lower bounded (see for instance [12]). We shall here consider only coupled tasks with fixed separation intervals as they apply to radar systems. There is a vast literature on this subject, treating offline and online cases and proposing various algorithms [19, 18, 17, 15, 11, 10, 9].

Let us now consider offline coupled task problems, where a set of tasks $\{a_j; L_j; b_j\}$ has to be scheduled on a single processor (the radar) with interleaving the coupled tasks but without overlapping the operations. In particular, the case $a_j = a$, $L_j = L$, $b_j = b$ for all j is called the identical coupled task problem.

Recently Brauner *et al.* [8] have described a new application of this coupled task model. They show that this problem is equivalent to a certain no-wait one-machine robotic cell problem usually studied in cyclic mode. For details on the equivalence, see [8]. In particular, producing a large number of single parts corresponds to the identical part production cycle. For this case, studying cyclic identical coupled tasks scheduling problems becomes a more relevant application.

The complexity of minimizing the makespan has been described by Orman and Potts [16]. Even the UET problem, $a_j = 1$, $L_j, b_j = 1$ for all j , is NP-hard and

Gerd Finke
Laboratoire G-SCOP
E-mail: Gerd.Finke@g-scop.inpg.fr

algorithms with worst-case performance ratio have been developed [1,4]. However, the complexity status of the identical coupled task problem remains open for both the non-cyclic case [16] and also for the cyclic case (see [2]).

Here, we consider the identical coupled task problem. Notice that the input of this problem is composed of four integers (three in the cyclic case): n , the number of tasks, a the duration of the first operation of a task, b the duration of the second operation and L the distance between both operations. Hence, it is a high-multiplicity scheduling problem [6,7] for which even proving that it belongs to NP might be difficult. Indeed, a description of a schedule (giving for instance the starting time of each of the n tasks) is not polynomial in the input size (which is, in our case, $\log n + \log a + \log b + \log L$). Ahr *et al.* [2] propose an algorithm linear in n but exponential in L . This algorithm has been adjusted to the cyclic case in [2] showing that the cyclic problem corresponds to finding the minimum mean cycle in a certain graph of 0-1 patterns (see also [14, 8]). This problem is polynomial in the size of the underlying graph [13] which, however in our case, has an exponential number of vertices. The computational experience in [8] shows, even with a significant reduction of the number of vertices, that for $a = 5$, $b = 3$ and $L = 41$, we have almost 9 000 vertices in the graph and the solution takes almost an hour computation time on a standard PC. Increasing L to 43 results in more than 14 000 vertices and we were not able to solve this instance with such an approach. In [3] it has been shown that for fixed a , b and L the optimal solution can be found in constant time. The constant is highly exponential in L and does not yield any practical computation. Note that one gets an $O(\log n)$ algorithm, including the input, which improves the $O(n)$ running time in [2].

We also want to mention that extensions of the identical coupled task problem turn out to be NP-hard. In [5] it is shown that the addition of strict precedences of identical coupled tasks $\{a = 1; L; b = 1\}$, *i.e.* ordered pairs of tasks are given that are not allowed to interleave, makes the problem NP-hard. Similarly, the problem $\{a; L; b\}$ becomes NP-hard if one adds a task-compatibility graph where two coupled tasks are compatible if they may interleave [20].

2 Feasible coupled task schedules

It is easy to construct feasible solutions for the cyclic identical coupled task problem with given integers a , b , L . We may assume that $a > b$. Take for instance $a = 5$, $b = 3$ and $L = 43$, a problem that could not be solved by the graph method. One can always pack, as long as possible, coupled tasks in succession without inserting unnecessary idle times. This would give the pattern $aaa \dots abbb \dots b$, which can then be repeated. For the example, there would be 9 a 's, an idle time of 3 units, followed by 9 b 's (which are separated by $a - b = 2$ units). We have placed 9 coupled tasks on the length 91 (the ratio is then $10\frac{1}{9}$).

The previous solution is working in both modes (cyclic and non cyclic). We shall give another solution that is rather surprising and very simple. It turns out that this is in fact an optimal solution. Note first that a cyclic solution is based on an infinite number of tasks. One is therefore looking for a feasible pattern of a 's and b 's, which can be attached identically and repeatedly to the left and the right to a copy of the given pattern. Thus in the cyclic mode there is no particular starting and finishing phase of the solution. The solution consists of using repeatedly the pattern ($a = 5$, idle = 0.5, $b = 3$). One may verify that this really results in a feasible schedule. In

this way, we have placed one coupled task in the cyclic sense on 8.5 units. The ratio is therefore also 8.5, improving the previous value. This solution is somewhat unexpected since the placement is partially at fractional starting times for the coupled tasks. It is interesting to notice that such fractional placements cannot be detected by the graph methods in [2, 8, 14, 3].

The main difficulty to solve the identical coupled task problem seems to be that one does not know enough about the structural properties of the optimal solution patterns.

2.1 Construction of solutions

Consider a cycle $\mathcal{C}$. Each coupled task on $\mathcal{C}$ can be described by its *window* $W = [a, L, b]$ giving the sequence of a -terms and b -terms that are placed in the L -section of W .

Notice that two consecutive b -terms are at least separated by $a - b$ time units. Equality with $(a - b)$ occurs in fact if the corresponding a -terms are consecutive without idle time. Therefore, the space utilization of the first b -term of two consecutive b -terms is at least a on the L -section of the window.

Based on this observation, we can count for the utilized space the number β of terms (ba) and the remaining number α of terms utilizing space a . These so-called profiles (α, β) , computed for each window of the cycle $\mathcal{C}$, are solutions of the constraint set

$$\begin{cases} L = \alpha a + \beta(a + b) + \gamma \\ \alpha, \beta, \gamma \geq 0 \end{cases} \quad (1)$$

where γ indicates the slack variable. It is somewhat remarkable that, for any feasible solution (α, β) of (1), one can construct cycles $\mathcal{C}(\alpha, \beta)$ so that every task window W of these cycles possesses the same profile (α, β) .

This construction can be done in polynomial time, based on simple divisions, formulas and a compact description of the cycle.

2.2 Optimal solution

We consider now the class $\mathcal{L}$ of cycles of the form $\mathcal{C}(\alpha, \beta)$. Using the constraint set (1), one can observe that the windows $[a, L, b]$ contain exactly $\alpha + 2\beta$ elements. It appears now quite natural to maximize $(\alpha + 2\beta)$, subject to the constraint set (1). In this way, one packs the most possible number of elements in each window.

Among the possible optimal solutions, one can describe the best one, (α^*, β^*) , so that the corresponding cycle $\mathcal{C}(\alpha^*, \beta^*)$ is the best in the class $\mathcal{L}$. Through rather lengthy technical arguments, one is then able to prove that the cycle $\mathcal{C}(\alpha^*, \beta^*)$ is in fact an optimal solution for the cyclic problem. Thus the cyclic identical coupled task problem has polynomial complexity.

References

1. A. A. Ageev and A. E. Baburin. Approximation algorithms for UET scheduling problems with exact delays. *Operations Research Letters*, 35(4):533 – 540, 2007.
2. D. Ahr, J. Békési, G. Galambos, M. Oswald, and G. Reinelt. An exact algorithm for scheduling identical coupled tasks. *Mathematical Methods of Operations Research (ZOR)*, 59:193–203, 2004.

3. Ph. Baptiste. A note on scheduling identical coupled tasks in constant time. Technical report, private communication, 2009.
4. J. Békési, G. Galambos, M. Oswald, and G. Reinelt. Improved analysis of an algorithm for the coupled task problem with UET jobs. *Operations Research Letters*, 37(2):93 – 96, 2009.
5. J. Blazewicz, K. Ecker, T. Kis, and M. Tanaš. A note on complexity of scheduling coupled tasks on a single processor. *Journal of the Brazilian Computer Society*, 7(3):23–26, 2001.
6. N. Brauner, Y. Crama, A. Grigoriev, and J. van de Klundert. A framework for the complexity of high-multiplicity scheduling problems. *Journal of Combinatorial Optimization*, 9:313–323, 2005.
7. N. Brauner, Y. Crama, A. Grigoriev, and J. van de Klundert. Multiplicity and complexity issues in contemporary production scheduling. *Statistica Neerlandica*, 61(1):75–91, 2007.
8. N. Brauner, G. Finke, V. Lehoux-Lebacque, C.N. Potts, and J. Whitehead. Scheduling of coupled tasks and one-machine no-wait robotic cells. *Computers and Operations Research*, 36(2):301–307, 2009.
9. C. Duron. *Ordonnancement en temps-réel des activités des radars*. PhD thesis, Université de Metz, 2002.
10. M. Elshafei, H. D. Sherali, and J. C. Smith. Radar pulse interleaving for multi-target tracking. *Naval Research Logistics*, 51:72–94, 2003.
11. A. Farina and P. Neri. Multitarget interleaved tracking for the phased radar array. *IEEE Proceedings F*, 27:312–318, 1980.
12. J. N. D. Gupta. Comparative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per job and time-lags. *Journal of Global Optimization*, 9:239–250, 1996.
13. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 1978.
14. V. Lebacque. *Théories et applications en ordonnancement : contraintes de ressources et tâches agrégées en catégories*. PhD thesis, Université Joseph Fourier, 2006.
15. D. J. Milojevic and B. M. Popovic. Improved algorithm for the interleaving of radar pulses. *IEEE Proceedings F*, 139:98–104, 1992.
16. A. J. Orman and C. N. Potts. On the complexity of coupled-task scheduling. *Discrete Applied Mathematics*, 72:141–154, 1997.
17. A. J. Orman, A. K. Shahani, and A. R. Moore. Modelling for the control of radar system. *Computers and OR*, 25:239–249, 1998.
18. A. K. Shahani, A. J. Orman, C. N. Potts, and A. R. Moore. Scheduling for a multifunction phased array radar system. *European Journal of Operational Research*, 90:13–25, 1996.
19. R. D. Shapiro. Scheduling coupled tasks. *Naval Research Logistics Quarterly*, 27(2):489–497, 1980.
20. G. Simonin, B. Darties, R. Giroudeau, and J.-C. König. Isomorphic coupled-task scheduling problem with compatibility constraints on a single processor. In *MISTA*, 2009.

A Hyperheuristic Approach to Belgian Nurse Rostering Problems

Burak Bilgin · Patrick De Causmaecker ·
Greet Vanden Berghe

1 Introduction

The nurse rostering problem involves the assignment of shifts to nurses with respect to several constraints like workload, legal restrictions, and contractual agreements [9]. The complexity of the problem attracts many researchers around the world. Since the regulations, requirements, and agreements differ from country to country, the models and solution methods differ as well [9].

The solution methods to tackle the nurse rostering problems are as diverse as the problem definitions. Estimation of distribution algorithm [1], constraint programming [6], case-based reasoning [2], and hybrid meta-heuristic methods [8] are examples of solution methods that cope with the nurse rostering problem. The success of metaheuristic methods depends on the weights of the objective function. Parr and Thompson propose a method for setting weights in an objective function for the nurse rostering problem in [15].

The nurse rostering problem is not the only scheduling problem encountered in hospitals. Beliën and Demeulemeester integrate nurse rostering and surgery scheduling in a single problem definition and address the resulting problem with a branch-and-price approach [3]. Gendreau et al. tackle the physician scheduling in emergency rooms in Canada using mathematical programming, tabu search, constraint programming and column generation [11].

This study is focused on the nurse rostering problems in Belgium. The instances of nurse rostering problems in Belgium are very diverse. Problem properties like workload, number of nurses, schedule periods, and contractual agreements vary among different institutions as well as among the wards within the same institution. Bilgin et al. propose

B. Bilgin, G. Vanden Berghe
KaHo Sint-Lieven, Information Technology
Gebroeders Desmetstraat 1, 9000 Gent, Belgium
E-mail: {Burak.Bilgin, Greet.VandenBerghe}@kahosl.be

P. De Causmaecker
K.U.Leuven Campus Kortrijk
Computer Science and Information Technology
Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: Patrick.DeCausmaecker@kuleuven-kortrijk.be

six neighbourhoods for the nurse rostering problems in Belgium [5]. Several neighbourhood sets are composed from these neighbourhoods and deployed within a variable neighbourhood search (VNS) algorithm.

Hyperheuristics are high level search strategies. The main defining property of hyperheuristics is that they operate on a set of heuristics instead of operating directly on the search space of the problem [7]. Hyperheuristics try to benefit from the strengths of each heuristic, and try to avoid their weaknesses. Improving hyperheuristics are iterative methods that select and execute a heuristic at each iteration and either accept or decline the new state of the candidate solution. Several heuristics and metaheuristics are deployed as the selection method and the acceptance criterion in such a setting [14].

Dowsland et al. combined simulated annealing as an acceptance criterion with tabu search as the selection method in a hyperheuristic [10]. The resulting hyperheuristic is applied to the problem of determining shipper sizes for storage and transportation. Kendall and Mohamad used great deluge as the acceptance criterion to tackle the channel assignment problem in cellular communication [12]. The selection method they have used is ‘simple random’. In ‘simple random’, heuristics are selected randomly and each heuristic has equal probability of getting selected at each iteration.

In this study, simulated annealing and great deluge acceptance criteria are deployed in hyperheuristics to tackle the problem. The solution methods are evaluated empirically on real world benchmarks. The performance of the methods are compared with the performance of the VNS algorithms presented in [5]. The simulated annealing hyperheuristic performs significantly better on most of the test cases.

The remainder of the paper is structured as follows: the problem definition and model are introduced in Section 2, the solution method is described in Section 3, experiments are presented in Section 4, and the paper is concluded in Section 5.

2 Problem Definition and Model

The problem addressed in this paper is initially introduced in [5], and it can be categorised as ASBI|RVNO|PLO according to the notation introduced in [13]. The search space of the problem is defined by the schedule period, skill types, shift types, and nurses. The parameters of each dimension of the search space are user-defined and they are variable between different problem instances. Shift types are defined by their start and end times, rest time before and after, and the effective job times.

Nurses are defined with their own set of skill types and individual contracts. Each nurse has a single primary skill type, and optionally a set of secondary skill types. Assignments are allowed only to one of the primary or secondary skill types of a nurse. Assignment of a shift to the primary skill type of the nurse is a soft constraint.

Several hard and soft constraints are applied to the problem. An overview of the constraints is presented in Table 1. The constraints defined in individual contracts are referred as horizontal constraints, and considered as soft constraints. The non-trivial constraints like horizontal and coverage constraints involve user parameterisation to a high degree. Most of the constraints that involve shift types are defined over a set of compatible shift types instead of a single shift type.

Horizontal constraints are organised around three generic types: counters, series, and successive series. *Counters* are defined over user-defined periods. They can have six subjects:

1. Shift types worked

Hard Constraints	Soft Constraints
Single Assignment Start Per Nurse Per Day	Coverage Constraints
No Overlap between Assignments	Assignment to the Primary Skill
Honour Skill Types	Rest Times
Operations on Defined Assignments Only	Requests
Schedule Locks	Horizontal Constraints

Table 1 Hard and soft constraints to the nurse rostering problem

2. Days worked
3. Days idle
4. Hours worked
5. Weekends worked
6. Weekends idle

If the subject is *shift types worked*, a set of compatible shift types are given as a parameter. The counters, except *weekends worked* and *weekends idle*, also have a set of day types parameters to define on which days the values need to be counted.

Series restrict the succession of occurrences of various subjects. *Series* can have five subjects, the same subjects as counters except the hours worked. Similar to counters, they have a set of compatible shift types given as a parameter if the subject is shift types worked.

Successive series restrict the succession of two series. They can have five subjects as the following combinations:

1. Shift types worked - shift types worked
2. Days worked - days idle
3. Days idle - days worked
4. Shift types worked - days idle
5. Days idle - shift types worked

The solution of the problem, the roster, consists of a set of assignments. An assignment is defined by a shift type that is assigned to a nurse on a particular day and for a particular skill type.

3 Solution Method

There are six problem-specific neighbourhoods defined in [5]: *assign shift*, *delete shift*, *single shift-day*, *change assignment based on compatible shift type*, *change assignment based on compatible skill type*, and *general assignment change*. A VNS algorithm is used to carry out a search in a steepest-descent fashion over these neighbourhoods [5].

In this study, a heuristic is defined to explore each neighbourhood. These heuristics check a fixed number of random solutions in the neighbourhood of the current solution and return the one with the best fitness value at each iteration. This strategy is referred to as tournament strategy. The number of solutions checked at each iteration is called the tournament factor.

The most promising neighbourhood set composition strategy in [5] is used to compose the heuristics sets in this work. If any of the nurses have secondary skill types,

the heuristic set is composed of *assign shift*, *delete shift*, *single shift-day*, *general assignment change*, and adds *change assignment based on compatible skill type*. If none of the nurses have a secondary skill type, *change assignment based on compatible skill type* heuristic is left out.

Improving hyperheuristics carry out two steps at each iteration: selecting a heuristic to apply to the candidate solution and accepting or rejecting the modified candidate solution. In this study, simple random is used as the heuristic selection method. Simulated annealing and great deluge methods are used as acceptance criteria.

The simulated annealing acceptance criterion accepts all improving and equal quality moves. The worsening moves are accepted with a probability that decreases throughout the execution. The formula of the probability at iteration $i + 1$ is presented in Equation 1. Here, f_i denotes the fitness value of the candidate solution at iteration i , t_r the remaining execution time, and t_e the total execution time.

$$p = \exp\left(-\frac{\frac{f_{i+1}-f_i}{f_i}}{\left(\frac{t_r}{t_e}\right)^2}\right) \quad (1)$$

Similar to simulated annealing, all improving and equal quality moves are accepted by great deluge. However, the worsening moves are accepted only if the fitness of the resulting solution is less than the deluge value. The value of this variable is decreased throughout the execution. The deluge value used in this study is presented in Equation 2. Here, f_0 denotes the fitness value of the candidate solution at iteration 0, t_r the remaining execution time, and t_e the total execution time.

$$d = f_0 * \left(\frac{t_r}{t_e}\right)^2 \quad (2)$$

4 Experiments

4.1 Experimental Settings

Experiments are carried out on the same data set and under the same conditions as in [5] to enable comparison between the results from both studies. The execution time is ten minutes for each run on normal and overload scenarios, and one minute for each run on absence scenario. The data set is collected from six different wards in two hospitals. It reflects the diversity of the problem properties among different wards and institutions.

Three different scenarios are considered for each data set: normal, overload, and absence. The overload scenario refers to a situation where the number of requested nurses is higher than the normal scenario. The absence scenario refers to rescheduling a roster due to an unforeseen absence of a nurse for an entire week. The input data, a sample output and penalty report from each ward-scenario couple are available in [4].

The algorithms are implemented in C#. MS Visual Studio 2005, Professional Edition is used for the experiments. The experiments are carried out on MS Windows Server 2003 Enterprise Edition SP 2 and Intel Pentium 4 CPU with 2.40 GHz and 2.00 GB of RAM.

Emergency	Normal		Overload		Absence	
	Average	Std. Dev.	Average	Std. Dev.	Average	Std. Dev.
VNS*	11234.05	178.80	26871.33	258.42	21175.67	19.12
SA-128	10922.50	199.16	26836.33	142.09	21146.67	0.00
SA-256	11038.50	316.31	26633.50	108.87	21146.67	0.00
GD-128	12087.17	765.41	27382.17	228.44	21149.17	2.64
GD-256	11501.83	354.71	26903.67	132.96	21151.17	9.26

Table 2 Results Emergency Ward

Geriatrics	Normal		Overload		Absence	
	Average	Std. Dev.	Average	Std. Dev.	Average	Std. Dev.
VNS*	5132.17	293.30	11484.50	398.61	8999.83	282.10
SA-128	3706.50	64.92	9686.33	57.76	8140.00	0.00
SA-256	3765.50	100.95	9647.67	38.64	8140.00	0.00
GD-128	3824.50	97.19	9915.00	80.52	8140.00	0.00
GD-256	3813.17	101.93	9820.50	81.02	8140.00	0.00

Table 3 Results Geriatrics Ward

Reception	Normal		Overload		Absence	
	Average	Std. Dev.	Average	Std. Dev.	Average	Std. Dev.
VNS*	22234.67	197.33	53414.17	228.00	28648.67	120.74
SA-128	21705.67	186.75	52893.17	136.32	28126.67	0.00
SA-256	21622.17	183.87	52981.67	74.05	28126.67	0.00
GD-128	21990.17	147.59	52889.67	123.07	28223.17	113.73
GD-256	21792.67	216.03	53119.17	218.94	28165.67	94.77

Table 4 Results Reception Ward

Meal Preparation	Normal		Overload		Absence	
	Average	Std. Dev.	Average	Std. Dev.	Average	Std. Dev.
VNS*	3018.80	36.10	10982.10	35.41	5448.33	116.53
SA-128	2994.70	39.29	11011.20	39.58	4925.17	49.38
SA-256	2967.73	45.32	10942.50	31.63	4912.33	38.31
GD-128	3117.13	27.26	11122.90	40.46	4954.17	48.72
GD-256	3089.03	55.50	11074.70	53.18	4978.17	53.23

Table 5 Results Meal Preparation Ward

4.2 Experimental Results

The objective of the experimentations is to compare the performance of the hyperheuristics and VNS. For this comparison, the results of the best performing VNS settings for each test case from [5] are used.

Since the best performing VNS settings vary among different ward-scenario couples, VNS variants are denoted with VNS* in Tables 2-7. Hyperheuristics are denoted with X-j in Tables 2-7. 'X' refers to the hyperheuristic type: SA for simulated annealing,

Psychiatry	Normal		Overload		Absence	
	Average	Std. Dev.	Average	Std. Dev.	Average	Std. Dev.
VNS*	8706.00	134.43	10643.00	65.50	12819.00	255.45
SA-128	8248.00	93.90	9995.00	90.95	11786.00	170.18
SA-256	8117.00	98.78	10021.00	127.06	11804.00	139.54
GD-128	8410.00	76.74	10223.00	204.56	12034.00	272.40
GD-256	8253.00	181.78	10196.00	180.26	12060.00	99.44

Table 6 Results Psychiatry Ward

Palliative Care	Normal		Overload		Absence	
	Average	Std. Dev.	Average	Std. Dev.	Average	Std. Dev.
VNS*	50201.25	1137.85	51217.50	685.19	56626.50	557.74
SA-128	48955.25	717.77	51109.00	932.06	55020.50	117.38
SA-256	48983.25	1063.56	50709.50	740.64	55008.50	228.98
GD-128	48438.50	682.60	51161.75	815.27	54997.50	191.93
GD-256	49677.00	1159.21	50786.00	681.97	55048.00	157.19

Table 7 Results Palliative Care Ward

GD for great deluge. The selection method used in the experiments is simple random. 'j' refers to the tournament factor of the heuristics.

The average fitness values over ten runs and their standard deviations are given in Tables 2-7. The Wilcoxon test has been used to detect statistically significant variances between the results. The confidence level is 95%. In Tables 2-7, data that are highlighted with bold characters denote the best results and the results that are not significantly worse than the best results.

The hyperheuristic algorithms performed significantly better than the VNS algorithms on 16 out of 18 problem instances. The VNS algorithms performed equally well as the hyperheuristics only on the palliative care and the meal preparation wards, both for overload scenarios. The simulated annealing hyperheuristic with tournament factor 256 was among the best performers on 17 out of 18 problem instances. The same hyperheuristic with tournament factor 128 was among the best performers on 15 out of 18 problem instances. The great deluge hyperheuristic with tournament factor 128 was among the best performers on 6 out of 18 problem instances, and the same hyperheuristic with tournament factor 256 on 5 out of 18 problem instances. Experimental results suggest that the simulated annealing hyperheuristic with tournament factor 256 is the most promising solution method for the problem.

5 Conclusions

Simulated annealing and great deluge hyperheuristics are used to tackle the nurse rostering problems in Belgian hospitals. A set of tournament heuristics are implemented on neighbourhoods proposed in [5]. Two different tournament factors, 128 and 256, are used during the experiments. The results of the best performing VNS methods to the problem are clearly improved by hyperheuristic methods on 16 out of 18 cases. For the remaining two cases hyperheuristic variants performed as good as the best performing VNS variant. The simulated annealing hyperheuristic with tournament factor 256 is the most promising solution method on overall.

Future work includes experimentation with further hyperheuristics. The relationships with the problem instance properties and the solution method performance will be studied. In this way, we aim for the selection of the most appropriate solution method and the parameter setting according to the problem input.

References

1. Aickelin, U., Burke, E., Li, J.: An estimation of distribution algorithm with intelligent local search for rule-based nurse rostering. *Journal of the Operational Research Society* **58**(12), 1574–1585 (2007)
2. Beddoe, G., Petrovic, S.: Enhancing case-based reasoning for personnel rostering with selected tabu search concepts. *Journal of the Operational Research Society* **58**(12), 1586–1598 (2007)
3. Beliën, J., Demeulemeester, E.: A branch-and-price approach for integrating nurse and surgery scheduling. *European journal of operational research* **189**(3), 652–668 (2008)
4. Bilgin, B.: Project web page of automation of nurse rostering in Belgian hospitals. <http://allserv.kahosl.be/~burak/project.html> (2008)
5. Bilgin, B., De Causmaecker, P., Rossie, B., Vanden Berghe, G.: Local search neighbourhoods to deal with a novel nurse rostering model. In: *The 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Montreal*, 20p (2008)
6. Bourdais, S., Galinier, P., Pesant, G.: HIBISCUS: A constraint programming application to staff scheduling in health care. *Lecture Notes in Computer Science* **2833**, 153–167 (2003)
7. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: *Handbook of Metaheuristics*, chap. 16: Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pp. 457–474. Springer New York (2003)
8. Burke, E.K., Curtois, T., Post, G., Qu, R., Veltman, B.: A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research* **188**(2), 330–341 (2008)
9. Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H.: The State of the Art of Nurse Rostering. *Journal of Scheduling* **7**(6), 441–499 (2004)
10. Dowsland, K.A., Soubeiga, E., Burke, E.: A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research* **179**(3), 759–774 (2007)
11. Gendreau, M., Ferland, J., Gendron, B., Hail, N., Jaumard, B., Lapierre, S., Pesant, G., Soriano, P.: Physician scheduling in emergency rooms. *Lecture Notes in Computer Science* **3867**, 53–66 (2007)
12. Kendall, G., Mohamad, M.: Channel assignment in cellular communication using a great deluge hyper-heuristic. In: *Proc. of the 2004 IEEE International Conference on Network (ICON2004)* (2004)
13. De Causmaecker, P.: Towards a reference model for timetabling and rostering. In: *The 7th International Conference on the Practice and Theory of Automated Timetabling, PATAT 2008, Montreal*, 10p (2008)
14. Özcan, E., Bilgin, B., Korkmaz, E.: A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis* **12**(1), 3–23 (2008)
15. Parr, D., Thompson, J.: Solving the multi-objective nurse scheduling problem with a weighted cost function. *Annals of Operations Research, Special Issue on Personnel Scheduling and Planning* **155**(1), 279–288 (2007)

Reactive Nurse Scheduling in Hospitals

Broos Maenhout · Mario Vanhoucke

1 Introduction

Human resources are one of the main causes for schedule disruptions in all kinds of companies and organizations. In the health sector, schedule disruptions caused by unplanned personnel absences have a dramatic effect on the budget for workforce staffing. In total 4% of the total resources spent on staffing are lost to schedule disruptions [1]. Since uncertainty is considerable in healthcare applications and these disruptions cannot be eliminated, decision support systems should be developed that adequately react to unexpected events or that workforce schedules are improved to be more responsive to disruptions. The adopted personnel policies and practices are important determinants for the motivation and work quality of employees [2]. Hence, in this context, constructing an appropriate nurse roster and/or being responsive to changes is essential for the delivery of care to patients.

2 Problem Description

Basically there are two different approaches to cope with disruptions in the personnel schedule such that the schedule remains valid in accordance with the schedule requirements (e.g. industry regulations, local laws), i.e. a reactive and a proactive scheduling approach. In reactive scheduling, a roster is revised and re-optimized after unexpected events have occurred. Hence, the scheduler waits for a disruption that renders the current optimal schedule invalid (i.e., tasks cannot be operated below a minimum number of required staff) and then applies a re-rostering method to rebuild the schedule. Often

Broos Maenhout

Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2,
9000 Gent (Belgium)

E-mail: Broos.Maenhout@Ugent.be

Mario Vanhoucke

Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2,
9000 Gent (Belgium)

Operations and Technology Management Centre, Vlerick Leuven Gent Management School,
Reep 1, 9000 Gent (Belgium)

E-mail: Mario.Vanhoucke@Ugent.be, Mario.Vanhoucke@Vlerick.be

times, the goal of re-rostering is to rebuild a new optimal schedule while minimizing the number of deviations to the original schedule. Applying traditional rostering methods produce new optimal schedules but often result in schedules that deviate considerably from the current schedules, which may not be very well accepted by the workforce. In the proactive scheduling approach, uncertainty in the scheduling environment is tackled by constructing a predictive schedule that accounts for statistical distributions to deal with uncertainty. The consideration of uncertainty information ahead of time is used to make the predictive schedule more robust. The degree to which schedules are made insensitive to disruptions can vary and different approaches can be adopted. However, a proactive technique will always require a reactive component to deal with schedule disruptions that cannot be absorbed by the baseline schedule. The number of interventions of the reactive component is typically inversely proportional to the robustness of the predictive baseline schedule.

Due to the importance of reactive scheduling in personnel rostering, we propose an optimization tool for the nurse re-rostering problem in which the current roster must be reconstructed with as few changes as possible. The scheduler must reassign other employees to cover the shift, while continuing to meet current demand and staffing requirements and time-related constraints guaranteeing the roster quality of the single personnel members.

3 Solution Procedure

In the literature, many successful exact and heuristic procedures were developed for the traditional nurse rostering problem that generates a workable baseline schedule, assuming complete information and a static and deterministic environment. An extensive overview of personnel shift scheduling problems can be found in [3] and [4] and for the nurse scheduling problem specifically in [5] and [6]. In contrast to the nurse rostering problem, the nurse re-rostering problem has received limited attention in the staff scheduling literature ([7], [8], [9], [10], [11]).

We focus on the application of established nurse rostering techniques to re-rostering problems and adapt these to meet the peculiarities (additional objectives and constraints) of re-rostering. The personnel scheduling optimization methods found in the literature provide excellent solutions to the nurse rostering problem. However, as uncertainty and absenteeism are inherent when dealing with personnel, these roster solutions may no longer be valid at certain point. We discuss how we deal with nurse re-rostering from an algorithmic point-of-view. We developed an exact branch-and-price procedure and a genetic algorithm, both incorporating problem-specific information concerning the nurse re-rostering problem. The one or the other method is applied to solve nurse re-rostering problem instances depending on the problem characteristics (objectives, constraints, problem size) and possible phase transitions in problem complexity.

4 Experimental Analysis

We provide computational and managerial insights into the nurse re-rostering problem. To provide a realistic environment to test our procedure, we obtained information on a real-life problem setting in a Belgian hospital about the structural organization of the

personnel scheduling process and data related to the required shift duties, staff mix, the nurse preferences, the nurse competencies, the shift scheduling policies and practices, nurse rosters, etc. Based on this roster information we constructed a problem set for the nurse re-rostering problem by specifying the occurrence of disruptions which are systematically varied in a controlled way. We validate the performance of the proposed procedures with respect to other techniques and the existing literature [10] and of the problem-specific operators for the nurse re-rostering problem. Moreover, we utilize the proposed procedure as a simulation tool and investigate the impact of different policy decisions and strategies on the resulting quality of the reconstructed nurse roster.

References

1. Ritchie K., Macdonald E., Gilmour W. and Murray K., Analysis of Sickness Absence Among Employees of Four NHS Trusts, *Occupational Environmental Medicine*, 56, 702-708 (1999)
2. F.F. Easton, D.F. Rossin and W.S. Borders, 1992, Analysis of alternative scheduling policies for hospital nurses, *Production and Operations Management*, 1, 159-174 (1992)
3. A.T. Ernst, H. Jiang, M. Krishnamoorthy, B. Owens and D. Sier, An Annotated Bibliography of Personnel Scheduling and Rostering, *Annals of Operations Research*, 127, 21-144 (2004)
4. A.T. Ernst, H. Jiang, M. Krishnamoorthy and D. Sier, Staff scheduling and rostering: A review of applications, methods and models, *European Journal of Operational Research*, 153, 3-27 (2004)
5. B. Cheang, H. Li, A. Lim and B. Rodrigues, Nurse rostering problems - a bibliographic survey, *European Journal of Operational Research*, 151, 447-460 (2003)
6. E.K. Burke, P. De Causmaecker, G. Vanden Berghe and H. Van Landeghem, The state of the art of nurse rostering, *Journal of Scheduling*, 7, 441-449 (2004)
7. M. Moz and M.V. Pato, An Integer Multicommodity Flow Model Applied to the Rerostering of Nurse Schedules, *Annals of Operations Research*, 119, 285-301 (2003)
8. M. Moz and M.V. Pato, Solving the Problem of Rerostering Nurse Schedules with Hard Constraints: New Multicommodity Flow Models, *Annals of Operations Research*, 128, 179-197 (2004)
9. M. Moz and M.V. Pato, A genetic algorithm approach to a nurse rerostering problem, *Computers & Operations Research*, 34, 667-691 (2007)
10. M. Moz and M.V. Pato, Solving a bi-objective nurse rerostering problem by using a utopic Pareto genetic heuristic, *Journal of Heuristics*, 14, 359-374 (2008)
11. Knighton S., An Optimal Network-Based Approach to Scheduling and Re-Rostering Continuous Heterogeneous Workforces, PhD Dissertation. Arizona State University, Tempe AZ (2005).

A multi-agent system handling personnel shortages in hospitals.

Stefaan Haspeslagh · Patrick De
Causmaecker · Greet Vanden Berghe

1 Introduction

Within the field of nurse rostering[2], solving personnel shortages is a central problem. Trivedi and Warner[11] and Gascon et al. [5] introduce a pool of float nurses. They solve shortages by using nurses from this pool. Siferd and Benton[10] tackle shortages by calling nurses from other wards and by allowing nurses to work overtime. Meisels and Kaplanski[6] pioneered a negotiation based approach. The latter authors introduce three stages. In stage 1, scheduling agents generate a local solution. In stage 2, a global feasible solution is constructed. The last stage is used by the agents to improve their local schedules. Meisels and Kaplanski assume a fixed pool of shared resources. Bard and Purnomo[1] propose various ways of tackling staff shortages. Relevant for the present discussion is the one in which initially negotiation takes place with a pool of float nurses. Consequently, nurses who are not needed by the wards, are placed on an on-call list. If none of the proposed solutions suffices the hospital can call in agency nurses.

Among Belgian Hospitals the idea of a mobile equipe has been introduced, similar to the pool of float nurses described above. This equipe, modeled as a virtual ward, contains nurses that are assigned to wards where shortages arise. There is a mobile team per group of wards of a department that require employees with more or less similar skills. The wards managers are supposed to negotiate the assignment of the float nurses to the actual wards, taking the personal preferences of the float nurses into account. The manager of the department assigns the nurses when the managers of a group cannot reach agreement and is responsible for rosters of the nurses of the mobile equipe. The schedule for the float nurses is known one day before and may only be changed

Stefaan Haspeslagh
KULeuven Campus Kortrijk
E-mail: stefaan.haspeslagh@kuleuven-kortrijk.be

Patrick De Causmaecker
KULeuven Campus Kortrijk
E-mail: patrick.decausmaecker@kuleuven-kortrijk.be

Greet Vanden Berghe
Katholieke Hogeschool Sint-Lieven
E-mail: greet.vandenbergh@kahosl.be

during a working day in case of extreme urgencies. For example, when due to a disaster, the emergency department suddenly needs significantly more personnel. The aim of this paper is to consider automated support for the negotiation described above. By explicitly modelling the negotiation process we hope to arrive at a transparent system with high acceptance among the negotiating parties, allowing to bring local concerns into the negotiation and providing an 'objective' decision making in a time-efficient procedure. We present a multi-agent system in which the agents representing the wards, negotiate on the allocation of the floating nurses. Several negotiation techniques exist to reach agreements in multi-agent environments. [7] argues that auctions are suitable within our setting. Section 2 discusses our choice of auction protocol. In section 3 we describe our model. Finally, we conclude and give directions for further research.

2 Auctions

We performed a comparative study and analysed a number of negotiation protocols details of which will be given in the presentation. We ultimately selected the ascending auction with package bidding as the most suitable protocol. This protocol has several advantages over the well-known Vickrey auction protocol[8]. We enumerate the most important issues for our setting.

First, Vickrey auctions are vulnerable to bidder collusion. Second, Vickrey auctions suffer from computational issues. In Vickrey auctions, each bidder submits bid on each of the $2^M - 1$ possible packages, with M the number of goods for sale. Even for relatively small M , an auction may become very complicated. Third, research has shown that Vickrey auctions are less suitable for auctioning interrelated items, e.g. in our setting requiring the same floating nurse to cover multiple shifts in the same ward.

The ascending auction protocol has been employed by the Federal Communications Commission in the US for its radio spectrum allocation, which shows similarities to the auction we want to set up. As bidders for the radio spectrum want to reserve adjacent frequencies, wards want to acquire services from the same nurse for several days/shifts, etc. We implemented an ascending auction protocol as proposed in [9].

3 Model

Every ward is represented by an agent. The agent is responsible for creating good working rosters for its ward nurses. Therefore, the agents use a metaheuristic algorithm based on the algorithm described in [4]. The agent of the mobile equipe ward is the auctioneer. The items for sale are the time units (a time unit is a certain shift on a particular day[3]) the nurses are available. The other wards in the group are the bidders. The packages on which they bid are a set of time units for which they require the services of a nurse of the mobile team. In each round of the auction, the agents can bid for one or more packages. In further rounds they can bid for new packages or they can raise their bids for previous wanted packages. The auction stops when there are two rounds with no new and no raised bids. The cost of a package, e.g. the cost of a bid is the total gain of the ward in case the package is accepted. The auctioneer can only accept bids for non over-lapping packages. Therefore, with every bid, the gain of sub-packages is given by the bidders. The auctioneer is then able to accept either the full package or a part of it. The revenue of the auctioneer is the value of the objective

function of its rosters decreased with the sum of the gain of the accepted packages. The goal of the auctioneer is to minimize its revenue. We implement the auction protocol from [9].

4 Conclusion and further research

With the setting described above, we deliver a proof of concept. We present the results of experiments with groups of 4 wards and one mobile equipe. Because of previous research and interviews with a large Belgian hospital we focus on real-world data instances. Details of the experiments will be given at the presentation. Extensions include design and implementation of other negotiation techniques and auction protocols, and study the cost of negotiation through comparison with central approaches.

References

1. J. Bard and H. Purnomo (2004), Real-time scheduling for nurses in response to demand fluctuations and personnel shortages, In Proceedings of the fifth International Conference in the Practice and Theory of Automated Timetabling, Pittsburgh, September 6-10, 6787.
2. E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem (2004), The State of the Art of Nurse Rostering. *Journal Of Scheduling*, 7(6), 441-499.
3. E.K. Burke, P. De Causmaecker, S. Petrovic, G. Vanden Berghe (2001), Fitness evaluation for nurse scheduling problems. In: Proceedings of the 2001 Congress on Evolutionary Computation.
4. E.K. Burke, T. Curtois, G. Post, R. Qu and B. Veltman (2008), A Hybrid Heuristic Ordering and Variable Neighbourhood Search for the Nurse Rostering Problem, *European Journal of Operational Research* 188(2), 330-341.
5. V. Gascon, S. Villeneuve, P. Michelon, and J. Ferland (2000), Scheduling the flying squad nurses of a hospital using a multi-objective programming model. *Annals of Operations Research* 96(1), 149-166.
6. E. Kaplansky and A. Meisels (to appear), Distributed personnel scheduling - negotiation among scheduling agents, *Annals of Operations Research*.
7. S. Kraus (2001), Automated Negotiation and Decision Making in Multiagent Environments, *Multi-Agent Systems and Applications*, 150-172.
8. L. M. Ausubel and P. R. Milgrom (2002), Package Bidding Vickrey vs. Ascending Auctions, *Revue conomique*, Vol. 53, No. 3, Dveloppement rcents de l'analyse conomique: Le congrs annuel de l'Association franaise de science conomique 2001, 391-402.
9. P. Milgrom (2000), Putting Auction Theory to Work: Ascending Auctions with Package Bidding, working paper.
10. S. Siferd and W. Benton (1992), Workforce staffing and scheduling: Hospital nursing specific models, *European Journal of Operations Research* 60(3), 233-246.
11. V. Trivedi and M. Warner (1976), A branch and bound algorithm for optimum allocation of float nurses, *Management Science* 22(9), 972-981.

Bi-Criteria Scheduling of an Outpatient Procedure Center

John W. Fowler • Serhat Gul • Brian T. Denton • Todd R. Huschka

A rising proportion of surgeries are performed on an outpatient basis. The demand for outpatient surgery typically occurs at the end of a stream of referrals which leads to considerable uncertainty in the type and number of surgeries to be scheduled on a particular day. Surgical services require coordination of many activities including surgical preparation, surgery, and patient recovery. Constructing surgery schedules that result in smooth patient flow is a complicated task due to the dependencies between these activities. The task is further complicated by the fact that the duration of each of these activities can be highly variable. Combining the high volume of activities in an outpatient surgical suite each day with uncertainty in the durations of activities presents challenging scheduling problems for outpatient clinic administrators.

Outpatient procedure clinics typically have multiple operating rooms, with a variety of supporting resources, such as nurses, nurse anesthetists, surgeons, intake rooms, recovery rooms, and various kinds of diagnostic equipment and surgical instrument kits. Surgical procedures occur in three major stages. Intake begins when the patient arrives on the day of surgery to initiate the check-in process and ends with the patient being taken to an operating room. Intra-operative care is defined as the time between when the patient reaches the OR bed and the time they are admitted to the recovery area. Recovery is the time between admission to the recovery area and the time that the patient is discharged from the recovery area. Even for very routine surgery (e.g. endoscopy) the time for intake, surgery, and recovery exhibits considerable uncertainty in duration.

There are a number of performance measures that outpatient clinic administrators may consider in relation to the efficiency of a surgical suite including: OR and recovery room utilization, surgeon and nurse idle time, patient waiting time, and late closure of the surgical suite. These measures are not independent of each other and often are in conflict with each other. For example, patient waiting time is generally negatively correlated with asset utilization

John W. Fowler
Arizona State University
john.fowler@asu.edu

Serhat Gul
Arizona State University
serhat.gul@asu.edu

Brian T. Denton
North Carolina State University
bdenton@ncsu.edu

Todd R. Huschka
Mayo Clinic
Huschka.Todd@mayo.edu

measures. Thus, the operation of a surgical suite is a multi-criteria problem. Furthermore, high fixed costs and the above mentioned factors combine to make for a highly constrained scheduling environment.

There is an extensive literature devoted to scheduling surgical services and we only include here a partial list of papers closely related to our research. Dexter et al. [1999], Dexter and Marcon [2006], Huschka et al. [2007], and Testi et al. [2007] evaluate scheduling heuristics for multiple ORs using discrete event simulation models. Denton et al. [2006] use a discrete event simulation model for the analysis and design of an endoscopy suite to investigate different surgeon-to-OR allocation scenarios. Lowery [1992], Lowery and Davis [1999], Marcon et al. [2003], and Tyler et al. [2003] use simulation to determine the number of resources (e.g. OR's and PACU beds) to support a given demand for surgical services. For a more extensive review of the literature on surgery planning and scheduling, the reader is referred to Magerlein and Martin [1978], Blake and Carter [1997], Gupta [2007], Gupta and Denton [2008].

In this presentation we discuss a number of heuristics, including a bi-criteria genetic algorithm, for scheduling a newly designed surgical suite at a major medical facility and evaluate the performance (patient waiting time and staff overtime) of the heuristics through the use of a discrete event simulation model. We discuss the unique design aspects of the suite, provide the details of the heuristics, give an overview of the simulation model, and present numerical results illustrating the trade-offs between the performance measures. In addition to discussing the performance of the heuristics, we also investigate the impact of daily surgical mix on the performance measures.

Acknowledgements

The first two authors were supported by National Science Foundation (NSF) grant DMI-0620504, and the third author was supported by NSF grant DMI-0620573. We are grateful to the staff of the Mayo Clinic OPC who was very helpful in supplying us with information for our model.

References

1. Blake, J.T., M.W. Carter. Surgical process scheduling: a structured review. *Journal of the Society for Health Systems*, 5(3), 17–30, 1997.
2. Denton, B., A. Rahman, H. Nelson, A. Bailey. Simulation of a multiple operating room surgical suite. In: Perrone, L., F. Wieland, J. Liu, B. Lawson, D.M. Nicol, R. Fujimoto (ed) *Proceedings of the 2006 Winter Simulation Conference*, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc, 414–424, 2006.
3. Dexter, F., A. Macario, R. Traub, M. Hopwood, D. Lubarsky. An operating room scheduling strategy to maximize the use of operating room block time: Computer simulation of patient scheduling and survey of patients' preferences for surgical waiting time. *Anesthesia and Analgesia*, 89, 7–20, 1999.
4. Dexter, F., E. Marcon. Impact of surgical sequencing on post anesthesia care unit staffing. *Health Care Management Science*, 9, 87–98, 2006.
5. Gupta, D. Surgical Suites' Operations Management. *Production and Operations Management*, 16, 689-700, 2007.
6. Gupta, D., B. Denton. Appointment scheduling in health care: Challenges and opportunities. *IIE Transactions*, 40, 800-819, 2008.
7. Huschka, T., B. Denton, S. Gul, J. Fowler. Bi-Criteria evaluation of an outpatient surgery procedure center via simulation. In: Henderson, S.G., B. Biller, M.H. Hsieh,

- J. Shortle, J.D. Tew, R.R. Barton (ed) *Proceedings of the 2007 Winter Simulation Conference*, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc, pp1510-1518, 2007
8. Lowery, J. Simulation of a hospital's surgical suite and critical care area. In: Swain, J., D. Goldsman, R. Crain, J. Wilson (ed) *Proceedings of the 1992 Winter Simulation Conference*, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc, 1071–1078, 1992.
9. Lowery, J., J. Davis. Determination of operating room requirements using simulation. In: Farrington, P., H. Nembhard, D. Sturrock, G. Ewans (ed) *Proceedings of the 1999 Winter Simulation Conference*, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc, 1568–1572, 1999.
10. Magerlein, J.M., J.B. Martin. Surgical demand scheduling: a review. *Health Services Research*, 13, 418–433, 1978.
11. Marcon, E., S. Kharraja, N. Smolski, B. Luquet, J. Viale. 2003. Determining the number of beds in the postanesthesia care unit: A computer simulation flow approach. *Anesthesia and Analgesia*, 96, 1415–1423, 2003.
12. Testi, A., E. Tanfani, G. Torre. A three-phase approach for operating theatre schedules. *Health Care Management Science*, 10, 163-172, 2007.
13. Tyler, D., C.A. Pasquariello, C.H. Chen. Determining optimum operating room utilization. *Anesthesia and Analgesia*, 96, 1114–1121, 2003.

Simulated Annealing Method for Maximizing Revenue on Parallel Machines

Jacek Juraszek · Erwin Pesch · Malgorzata Sterna

1 Introduction

In every real company the main goal is to achieve the highest possible income. As a matter of fact, the problem boils down to the revenue maximization case, which might be defined as allocating resources in acceptable time windows to realize some customer tasks in order to retrieve payment. The main difference between the problem investigate the scheduling problem rather than the "classical" scheduling theory (cf. e.g. [?]) is that in the latter case tasks provided by clients could not be left undone. However, real-world companies have limited resources and if the stream of clients requests exceeds production capabilities of such a company, then some requests have to be rejected as non-profitable. Moreover, we investigate the scheduling problem rather than the revenue management one (cf. e.g. [?]), because all parameters are fixed in advance and they cannot be negotiated in order to maximize the company income.

We consider a set of client orders, whose realizations, within feasible time windows, give certain revenue for a producer. The production capabilities are limited and the producer has to decide which orders to schedule and which ones are not profitable enough to accept them. There is also defined feasible delay for each order, but every delay in completing an accepted order causes the penalty decreasing the total revenue. This penalty usually depends on due date involving criteria (cf. e.g. [?]) such as for example on the tardiness or lateness. The revenue maximization problem has been studied for the single machine environment so far (cf. e.g. [?]). In our research, we focused on the parallel machine case, proposing three algorithms, which were evaluated in the extended computational experiments.

Jacek Juraszek
Institute of Computing Science, Poznan University of Technology
E-mail: jacek.juraszek@cs.put.poznan.pl

Erwin Pesch
Institute of Information Systems, University of Siegen
E-mail: erwin.pesch@uni-siegen.de

Malgorzata Sterna
Institute of Computing Science, Poznan University of Technology
E-mail: malgorzata.sterna@cs.put.poznan.pl

2 Problem formulation

The analyzed problem concerns scheduling a set of jobs $\mathcal{J} = \{J_1, J_2, \dots, J_n\}$ on the set of parallel identical machines $\mathcal{M} = \{M_1, M_2, \dots, M_m\}$. Processing a job on a machine cannot be preempted. Each job is described by the release time r_j , when a job is considered to be ready for executing, and the processing time p_j , which defines the consumption of the system resources. The decision maker has an option to reject some jobs without any additional penalty to the total revenue value, but the rejected jobs give obviously no income. All accepted jobs have to be completed before their deadlines D_j . There is also due date d_j defined for each job, as the moment from which the penalty starts reducing the maximum possible job revenue q_j . The income from completing a job is defined as the given revenue of this task decreased by its weighted tardiness $w_j * \max\{C_j - d_j, 0\}$. The decreased revenue represents the occurrence of the fine paid for acceptable delays, when the job completion time exceeds the due date, but it still does not exceed the deadline.

The formulated scheduling problem is NP-hard. Its intractability results from the NP-hardness of the order acceptance problem on the single machine ([?], [?]). Moreover, the optimal solution maximizing the total revenue requires minimizing the weighted tardiness, which is an intractable problem even for the single machine ([?]).

3 Solution methods

Every method solving the considered problem has to answer three questions on particular jobs. First, it has to decide whether a job is accepted and will be executed in the system. After that the method has to assign the job to a machine and, finally, to schedule it on this machine. Due to the NP-hardness of the analyzed problem, we designed three different methods, each answering all mentioned questions but in different fashion: the branch and bound algorithm (B&B), the list scheduling heuristic (LA) and the simulated annealing metaheuristic (SA).

The branch and bound algorithm constructs an optimal solution by analyzing all partitions of a set of jobs to two subsets of accepted and rejected ones. Then, it investigates all partitions of the accepted jobs to machines. For a certain assignment of jobs, B&B checks all feasible permutations of jobs on machines. Constructing a partial schedule is suppressed, if the current revenue increased by the total revenue which might be obtained from non-assigned jobs (i.e. the upper bound of the criterion value) does not exceed the lower bound for the total revenue achieved so far.

The list scheduling algorithm (cf. e.g. [?]) was used as an auxiliary procedure for B&B and SA and as the source of reference solutions in computational experiments. At each iteration, LA assigns an available job to a free machine, if such an assignment is feasible with regard to the job's deadline and if it will increase the overall revenue. Not assigned jobs are meant to be rejected. The method iterates over the set of jobs according to the priority dispatching rule (cf. e.g. [?]). LA was developed in order to obtain an initial solution for SA and the initial lower bound for B&B.

The simulated annealing algorithm implemented within the research is based on the classical SA skeleton (cf. e.g. [?]) extended with the local search procedure. The SA solution is represented as an assignment of jobs to machines. The set of machines is extended with an additional dummy machine, which collects rejected jobs. Such a solution is transformed to a schedule by an exact approach determining the optimal

sequence of jobs on machines (excluding the dummy machine representing the set of rejected jobs). We proposed two move definitions, generating new solutions based on shifting or interchanging jobs between two assignment sets, i.e. two machines. Every new solution found by the algorithm is improved with local search, i.e. by the efficient embedded simulated annealing method. The cooling scheme is based on the geometrical temperature reduction. The initial temperature was determined by a tuning process. The SA method terminates after a given number of diversifications and iterations without improvement.

4 Computational experiments

Computational experiments were performed for 4 groups of randomly generated instances, which differ by the distribution of job time parameters. There were instances with dense and diffused release times, i.e. close or distant to/from the beginning of a schedule. Similarly there were instances with "tight" and "loose" due dates and deadlines, in terms of the length of the interval between the release time and these two time parameters.

Test results showed the strong dependency of the effectiveness of the heuristic methods on the input data. Due to the huge time complexity B&B could not be applied for large instances of the problem, but its efficiency was high enough to provide us the tool for rating non-optimal methods for smaller instances. For most of those small instances, SA computed optimal solutions, constructing schedules with 98.73% of the optimal revenue value on average. SA results were of almost 9.26% better than LA results. For large instances SA improved the initial revenue value by 9.96% on average.

5 Conclusions

In this paper we presented the problem of accepting and scheduling non-preemptive jobs on identical parallel machines in order to maximize the total revenue. Due to its NP-hardness, we implemented the metaheuristic algorithm and compared it to the optimal B&B in the computational experiments. The research was inspired by the real world production environment ([?]), which is the subject of our further research.

References

1. Asbach L, Dorndorf U, Pesch E, Analysis, modeling and solution of the concrete delivery problem, *European Journal of Operational Research*, 193, 820–835 (2009)
2. Blazewicz J, Ecker K, Pesch E, Schmidt G, Weglarz J, *Handbook on scheduling: from theory to applications*, Springer, Berlin-Heidelberg-New York (2007)
3. Haupt R, A survey of priority rule-based scheduling, *OR Spektrum*, 11, 3–16 (1989)
4. Kirkpatrick S, Gelatt CD, Vecchi MP, Optimization by simulated annealing, *Science*, 220(4598), 671–680 (1983)
5. Lawler EL, A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness, *Annals of Discrete Mathematics*, 1, 331–342 (1977)
6. Lewis HF, Slotnick SA, Multi-period job selection: planning work loads to maximize profit, *Computers and Operations Research*, 29, 1081–1098 (2002)
7. McGill JI, van Ryzin GJ, Revenue management. Research overview and prospects, *Transportation Science*, 33, 233–256 (1999)
8. Slotnick SA, Morton TE, Order acceptance with weighted tardiness, *Computers and Operations Research*, 34, 3029–3042 (2007)

Scatter search for a weighted tardiness flow shop problem

Wojciech Bożejko · Mariusz Uchroński ·
Mieczysław Wodecki

Extended abstract

This work concerns the artificial intelligence instruments application in the building schedule. In this article an original optimization scatter search algorithm was presented, taking into consideration both technological and organizational restrictions. The algorithm was applied to the real analysis of the industrial building project realization.

The problems of tasks' scheduling with sum-cost goal functions have a long, over thirty years history. A great number of works concern scheduling on one machine. For multi-machine problems research more often involves the issue of delay tasks number minimization, i.e. $\sum U_i$ (where $U_i = 1$, if $C_{i,m} > d_i$, otherwise $U_i = 0$). In the work of Lenstra et al. [5] it is proved that for two machines the problem $F2||\sum U_i$ is strongly NP-hard. To the most important work, containing main theoretical results as well as good approximation algorithms belongs Bulfin and Hallah [1].

For a problem with many machines $F||\sum U_i$ Hairi and Potts [3] described an optimal algorithm based on the division and restriction method, by means of which in a reasonable time instances with 25 tasks and 3 machines can be solved. Genetic algorithms presented in works of Ucar and Tasgetiren [10] are ones of a few metaheuristic algorithms which have been made exclusively to the solve multi-machine flow-shop problem with a criterion $\sum w_i U_i$. There are considerably less works devoted to problems of costs delay sum minimization, i.e. the function $\sum w_i T_i$ and its particular example $\sum T_i$. The difficulty of this problem is verified by the fact that for the simplest example of two machines and goal function $\sum T_i$ the newest exact algorithms (Schaller [8]) solve instances to 20 tasks. Whereas, for the most difficult (discussed in this chapter) problem $F||\sum w_i T_i$ only a few works have been published describing optimal algorithms (the newest one - Chung et al. [2]). A wide range of methods, algorithms and publications covering multi-machine problems of tasks scheduling with costs delay sum minimization is included in works of and Valada et al. [11].

Wojciech Bożejko
Wrocław University of Technology
Institute of Computer Engineering, Control and Robotics
Janiszewskiego 11-17, 50-372 Wrocław, Poland, E-mail: wojciech.bozejko@pwr.wroc.pl

Mariusz Uchroński
Wrocław University of Technology
Institute of Computer Engineering, Control and Robotics
Janiszewskiego 11-17, 50-372 Wrocław, Poland, E-mail: mariusz.uchronski@pwr.wroc.pl

Mieczysław Wodecki
University of Wrocław
Institute of Computer Science
Joliot-Curie 15, 50-383 Wrocław, Poland, E-mail: mwd@ii.uni.wroc.pl

The Total Weighted Tardiness Flow Shop Problem (in brief TWTFPS) can be defined in the following way: here is a set of n tasks $\mathcal{J} = \{1, 2, \dots, n\}$ which are to be performed by means of m machines from the set $\mathcal{P} = \{1, 2, \dots, m\}$. The task $j \in \mathcal{J}$ should be performed consecutively on each machine, but the task performance on a k machine (operation O_{jk}) can start only after the completion of this task performance on a machine $k - 1$ ($k = 2, 3, \dots, m$) and the sequence of tasks performance on each machine must be the same. Furthermore, in any moment the machine can perform at most one task and the task performance cannot be broken. For the task $i \in \mathcal{J}$, let p_{ij} be the time of performance on the machine $j \in \mathcal{P}$, d_i the required completion time, and w_i the cost function weight.

For the set tasks sequence by $C_{i,j}$ we mark the completion time of the task $i \in \mathcal{J}$ on a machine $j \in \mathcal{P}$. Then, $T_i = \max\{0, C_{i,m} - d_i\}$ is a tardiness, and $f_i(C_{i,m}) = w_i \cdot T_i$ cost of tardiness of task performance. It should be determined such a sequence of tasks performance that it would minimize the costs tardiness sum i.e. the sum $\sum_{i=1}^n f_i(C_{i,m}) = \sum_{i=1}^n w_i \cdot T_i$.

The scatter-search method. The main ideas of this method are presented also in the work of James et al. [4]. In the proposed algorithm the base of the procedure operation, which generates the path linking two selected solutions is the scheme of multi-step fusion *MSXF* an operator programmed in the work of Reeves and Yamada [7]. As a measure of distance between solutions (permutations) serving to manage the process of making the path between $\pi(i)$ and $\sigma(i)$ the Hamming's measure was adopted: $H(\pi, \sigma) = \text{number of } i \text{ such that } \pi(i) \neq \sigma(i)$. The calculation complexity of determining the distance value is linear $O(n)$, where n is the number of elements in a permutation. In the path-relinking procedure each next path element is determined by looking through the neighborhood, i.e. some subset of solutions space generated by insert moves.

Moves and neighborhood. Let $\pi = (\pi(1), \dots, \pi(n))$ be a n element permutation of the Φ_n set.

The *insert move* i_l^k presents an element $\pi(k)$ (from the k position in π) on the position l , generating a permutation $i_l^k(\pi) = \pi_l^k$. If $l \geq k$, then

$$\pi_l^k(i) = \begin{cases} \pi(i), & \text{if } i < k \vee i > l, \\ \pi(i+1), & \text{if } k \leq i < l, \\ \pi(k), & \text{if } i = l. \end{cases}$$

and similarly in the case when $k > l$. The insert move we will call in brief *i-move*. Its calculation complexity equals $O(1)$. There are $n(n-1)$ of all such moves.

The permutation neighborhood $\pi \in \Phi_n$ generated by all *i-moves*:

$$\mathcal{N}(\pi) = \{i_l^k(\pi) : l, k = 1, 2, \dots, n \text{ and } l \neq k\}. \quad (1)$$

The sequences of directly succeeding jobs $\pi^B = (\pi(a), \pi(a+1), \dots, \pi(b-1), \pi(b))$, ($1 \leq a < b \leq n$) in permutation $\pi \in \Phi_n$ we call subpermutations. A *weight* of a subpermutation $\mathcal{WT}(\pi^B) = \sum_{i=a}^b w_{\pi(i)} \cdot T_{\pi(i)}$, and a *length* $\mathcal{L}(\pi^B) = C_{\pi(b),m}$. Let $\Phi_n(\pi^B) \subseteq \Phi_n$ be a set of permutations which can be generated from a permutation π by swapping an order of elements in a subpermutation π^B . The number of elements of the set $\Phi_n(\pi^B)$ equals $(b-a+1)!$.

Let $\mathcal{F}$ mean one of the defined above goal functions, i.e. $\mathcal{WT}$ (a weight) or $\mathcal{L}$ (a length). For a subpermutation π^B let

$$F^{mi}(\pi^B) \leq \min\{\mathcal{F}(\delta) : \delta \in \Phi_n(\pi^B)\},$$

$$F^{mx}(\pi^B) \geq \max\{\mathcal{F}(\delta) : \delta \in \Phi_n(\pi^B)\}.$$

For any permutation $\gamma \in \Phi_n(\pi^B)$ a value of $F^{mi}(\pi^B)$ and $F^{mx}(\pi^B)$ are lower and upper bound of $\mathcal{F}(\gamma)$, therefore $\mathcal{F}(\gamma) \in [F^{mi}(\pi^B), F^{mx}(\pi^B)]$. Subpermutations π^B we call θ -optimal ($0 \leq \theta \leq 1$) due to the function F , if

$$F(\pi^B) \in [F^{mi}(\pi^B), \theta(F^{mx}(\pi^B) - F^{mi}(\pi^B))].$$

In any permutation we will determine two types of subpermutations called blocks in the further part of the paper: (1) $\mathcal{T}$ -blocks, consist of on-time jobs, (2) $\mathcal{D}$ -blocks consist of late jobs.

Blocks of on-time jobs. Subpermutation $\pi^T = (\pi(a), \pi(a+1), \dots, \pi(b))$ is a $\mathcal{T}$ -block in a permutation π , (a) each job from π^T is on-time in the permutation π , so $\forall j \in \{a, a+1, \dots, b\}$, $C_{\pi(j),m} \leq d_{\pi(j)}$, (b) π^T is θ -optimal due to the length $\mathcal{L}$. Because each job in a $\mathcal{T}$ -block is on-time, therefore the cost of its executing equals 0. Then, π^T is an optimal schedule due to the cost.

Blocks of late jobs. Subpermutation of jobs $\pi^D = (\pi(a), \pi(a+1), \dots, \pi(b))$ is a $\mathcal{D}$ -block in the permutation $\pi \in \Phi_n$, if: (a') any job $\pi(j)$ ($j = a+1, a+2, \dots, b$) moved into the first position in π^D (i.e. after executing an *insert* move i_a^j) is late, (b') due to the cost a subpermutation π^D is θ -optimal.

From the constraint (a') it follows that each job from a $\mathcal{D}$ -block, after moving into the first position in π^D , is late. Algorithms of blocks determination in a permutation are described in the paper of Wodecki [12]. The number of blocks and the number of elements in each blocks depend on the parameter θ ($0 < \theta < 1$).

Let $\mathcal{B} = \{\pi^1, \pi^2, \dots, \pi^t\}$ be a set of blocks (i.e. $\mathcal{T}$ and $\mathcal{D}$ blocks) in the permutation π and let $\mathcal{N}(\pi)$ be the neighborhood generated by insert moves (see (1)). We are applying restricted neighborhoods in algorithms. We are eliminating permutations generated from π by changing an order of elements in any block. Therefore, we take $\mathcal{N}(\pi) = \mathcal{N}(\pi) \setminus \bigcup_{\delta \in \mathcal{B}} \Phi_n(\pi^\delta)$.

Computational experiments. To conduct the statistical analysis of the algorithm operation the method was also tested on test data. Results were compared with the NEH (Nawaz et al. [6]) algorithm. In Taillard's work [9] there was presented the commonly used these days method of data generation to evaluate approximation algorithms of solution of different combinatorial optimization problems. In their primary version it enables to find instances for the flow shop problem only on the based of the criterion $C_{\max}$. Applying this method the times to conduct the operation were generated. An integer weight was generated from the uniform distribution [1,10]. The required completion times were generated using the following procedure:

Step 1: Calculate $P_i = \sum_{j=1}^m p_{i,j}$, $i = 1, 2, \dots, n$, the total time of carrying out the task $i \in \mathcal{J}$ on machines.

Step 2: For each task $i \in \mathcal{J}$ determine the required completion time: $d_i = P_i \cdot (1 + \mu \cdot 3)$.

The parameter μ is the random variable realization of a uniform schedule on the $[0,1]$ distance. This generator is described in Taillard's work [9]. The generated data form 8 groups with a size nxm: 20x5 20x10, 50x5, 50x10, 50x20, 100x5, 100x10 and 100x20. For each size 25 examples were determined, so all data embraced 200 examples.

The scatter search algorithm was coded in C++ language. The calculation experiments were performed on the IBM microcomputer with a 1.8 GHz clock. To determine the starting solution the NEH algorithm was employed. In literature, there are neither data nor results to compare with this problem. Therefore, an additional series of instances in a small size nxm was generated: 8x4, 8x6, 10x4, 10x6, 12x4, 12x6 and 12x8 -

10 examples for each size. The obtained results for the data of the NEH algorithm were compared with optimal values determined by a total search. The average relative error equals 8.62%. Before starting the basic test to determine the algorithm parameters calculations were made on a small data group - one example from each size. On this basis it was assumed that the parameter θ (in the block definition) would be equal to 0.2.

Due to the obtained results it can be stated that the blocks application brought about a considerable improvement. The algorithm with blocks determines better solutions in a shorter time. The average improvement of the starting solution (determined by NEH) equals 6.20%; whereas for the algorithm without blocks equals 4.55%. Furthermore, the average time of this algorithm operation is about 22% smaller. The average calculation times of one instance equals 7.25 and 9.13 seconds, respectively. Shortening of the time calculation results from the fact that in case of the algorithm without blocks the whole neighborhood is searched. Number of its elements is $O(n^2)$ and the calculation time of the goal function value for one element is $O(nm)$. In turn, the procedure of blocks determination in a permutation has a calculation complexity $O(n^2m)$. Besides, the calculation experiments proved that restricted neighborhood, which is generated with the use of blocks elimination properties, is about 35% smaller than a whole neighborhood. Therefore, elimination, through filtration (heuristic indirect search) of many non-perspective elements, causes values improvement of fixed solutions and the calculation time shortening.

References

1. Bulfin R.L., Hallah R., Minimizing the weighted number of tardy jobs on two-machine flow shop, *Computers&Operations Research*, 30, 1887-1900 (2003).
2. Chung C., Flynn J., Kirca O., A branch and bound algorithm to minimize the total tardiness for n-machine permutation flowshop problems, *European Journal of Operational Research*, 174(1), 1-10 (2006).
3. Hariri A.M.A., Potts C.N., Branch and Bound Algorithm to Minimize the Number of Late Jobs in a Permutation Flowshop, *Eur. Jour. of Oper. Res.*, 38, 228-237 (1989).
4. James T., Rego C., Glover F., Sequential and Parallel Path-Relinking Algorithms for the Quadratic Assignment Problem, *IEEE Intelligent Systems*, 58-65 (2005).
5. Lenstra J.K., Rinnoy Kan A.G.H., Brucker P., Complexity of Machine Scheduling Problems, *Annals of Discrete Mathematics*, 1, 343-362 (1977).
6. Nawaz M., Ensore E.E., Ham I., A heuristic algorithm for the m -machine, n -job flow-shop sequencing problem, *OMEGA*, 11/1, 91-95 (1983).
7. Reeves C. R., Yamada T., Solving the Csum Permutation Flowshop Scheduling Problem by Genetic Local Search, *IEEE International Conference on Evolutionary Computation*, 230-234 (1998).
8. Schaller J., Note on minimizing total tardiness in a two-machine flowshop, *Computers&Operations Research*, 30(5), 3273-3281 (2005).
9. Taillard E., Benchmarks for basic scheduling problems, *European Journal of Operational Research*, 64, 278-285 (1993).
10. Ucar H., Tasgetiren M.F., A particle swarm optimization algorithm for permutation flow shop sequencing problem with the number of tardy jobs criterion, *Proceedings of 5th International Symposium on Intelligent Manufacturing Systems*, 1100-1120 (2006).
11. Valada E., Ruiz R., Minella G., Minimizing Total Tardiness in the m- Machine Flow-shop Problem: A Review and Evaluation of Heuristics and Metaheuristics, *Computers and Operations Research*, Vol. 35, No. 41350-1373 (2006).
12. Wodecki M., A block approach to earliness-tardiness scheduling problems, *Advanced Manufacturing Technology*, Vol. 40, No. 7-8, 797-807 (2008).

A VNS-based solution approach for a flexible flow shop problem with limited buffer

Christian Almeder · Stefan Katzensteiner ·
Richard F. Hartl

1 Problem description

In this work we consider a real-world scheduling problem of the metal-working industry. The production process consists of several steps, which can be grouped in two stages. The first stage is arranged as an assembly line, where the raw material will be transformed into a semifinished product. Except of major changes of the product type, the process runs continuously and new semifinished products arrive according to a predefined, product-specific rate. The second stage, the post-processing step, consists of quality control, repairing, and finishing the product. These steps are performed on different parallel stations, but since the products are difficult to handle because of their size and weight, there is a preferable link between the stations. Only in an extraordinary case such as machine breakdown or similar problems, products would take different ways through the shop floor. Hence all steps of the second stage can be simplified to a parallel machine case.

Due to the considerably varying processing times of the quality control and repairing process, there is a small buffer between the first and the second stage, which is necessary to avoid suspending the first stage in case of problems in the second stage. The buffer has a small capacity of one to two hours output of the first stage, but it gives the opportunity to reorder and assign the products to the parallel machines of the second stage. Figure 1 presents the principle problem layout and the points where it is possible to schedule respectively reschedule the products.

According to [1], this problem would be a flexible flow shop problem with limited buffers, but with two special properties:

- The two machines in the second stage are unrelated.
- There is one common buffer for all parallel machines.

The objective function for measuring the performance of a schedule is the total flow time, so the whole problem may be denoted as $FF2|block|\sum_i C_i$.

Christian Almeder, Stefan Katzensteiner, Richard F. Hartl
University of Vienna, Dept. of Business Studies, Brünnerstr. 72, A-1210 Vienna, Austria
E-mail: Christian.Almeder@univie.ac.at, Stefan.Katzensteiner@univie.ac.at,
Richard.Hartl@univie.ac.at

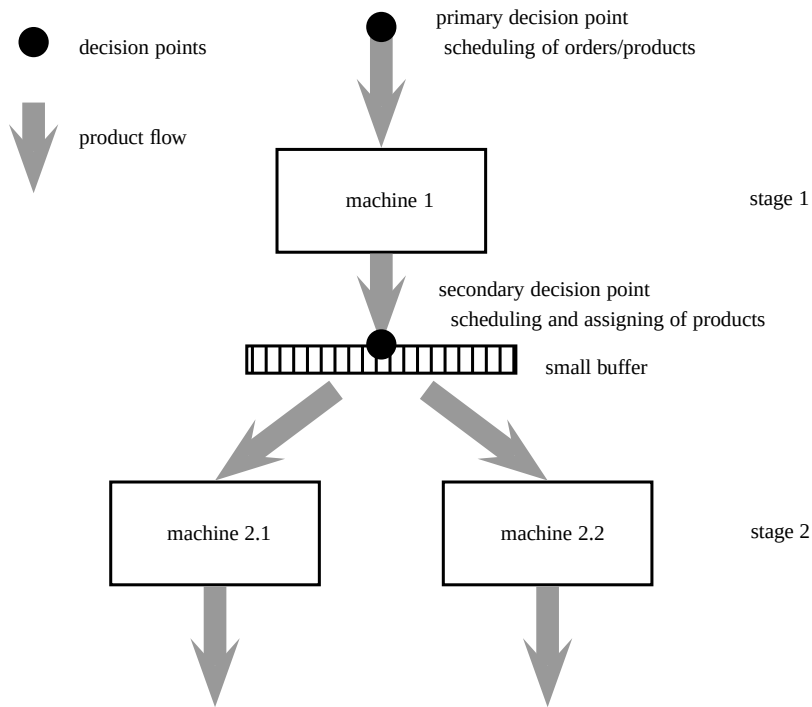


Fig. 1 Problem layout and points for scheduling and rescheduling

2 Method

To solve this problem we have to determine a job sequence for the first stage and a job sequence and assignment for the second stage. The sequence on the first stage has definitely the larger impact on the solution, especially because of the limited buffer. So similar to [2] we only determine the sequence of jobs for the first stage. For the second stage a simple first-in-first-out (FIFO) rule will be applied, i.e. each job leaving the first stage will be assigned to the end of the queue in the buffer and each time a machine of the second stage gets available the first job in the buffer will be assigned to that machine.

As a solution approach we developed a variable neighborhood search (VNS) respectively a variable neighborhood descent (VND) method (cf. [3]). We use three different kind of neighborhoods for this scheduling problem (cf. [4]): *transpose* (swapping two adjacent jobs), *insert* (remove a job from the schedule and insert it somewhere else), and *interchange* (swap any two jobs in the sequence).

Testing different variation of algorithms on several test instances have shown that the following solution algorithms work best:

VND — a variable neighborhood descent using all three neighborhoods in the order given above.

Table 1 Comparison of different solution approaches.

buffer	SPT-S1	SPT-S1+S2	SPT-S2	VND	VNS-best	VNS-first
20%	100.00%	103.94%	141.97%	95.60%	96.01%	95.92%
10%	100.00%	103.41%	139.36%	95.22%	95.48%	95.42%
2%	100.00%	99.00%	130.62%	89.65%	90.89%	90.58%
Total	100.00%	102.12%	137.32%	93.49%	94.13%	93.97%

VNS-first — a variable neighborhood search using the VND as a local search method with a first-improvement strategy; the neighborhoods for the shaking step consist of multiple application of the above neighborhoods.

VNS-best — similar to VNS-first, but with a best-improvement strategy.

In this first investigation we consider the total waiting time. For a single machine problem the shortest processing time (SPT) would be the optimal solution. Hence, we include in our tests 3 different versions of the SPT rule as benchmark results:

SPT-S1 The SPT sequence is based on the processing time of the first stage

SPT-S1+S2 The SPT sequence is based on the processing time of the first and the average processing time of the second stage.

SPT-S2 The SPT sequence is based solely on the minimal processing time of the second stage

3 Results and Conclusions

For testing we used in total 150 test instances with three different buffer sizes: 2%, 10%, and 20% of the total number of jobs. Table 1 depicts a small portion of these results showing the impact of the buffer size on the solution quality. The SPT-S1 method is used as benchmark results. It shows clearly that among the SPT-methods overall the SPT-S1 leads to the best results although the combined version SPT-S1+S2 is slightly better if the buffer size is small. SPT-S2 is definitely not a good choice for this kind of problems.

All three VNS/VND methods deliver considerably better results (all using the same computational time). Overall the solutions of SPT-S1 are improved by around 6%. There is a clear trend that with a smaller buffer size the difference between the dispatching-based solution and the metaheuristic approach increases and improvements of up to 10% are possible.

References

1. M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd edn. (Springer, 2008)
2. J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse, F. Werner, *Computers & Operations Research* **36**, 358 (2009)
3. P. Hansen, N. Mladenovic, *European Journal of Operational Research* **130**, 449 (2001)
4. M. den Besten, T. Sttze, in *Proceedings of the 4th Metaheuristic International Conference MIC2001* (2001), pp. 545–549

Mathematical Modeling for Maximising Gate Receipt Problem

Nor Hayati Abdul Hamid • Graham Kendall • Mujgan Sagir

1 Introduction

Malaysian football is seeing a decrease in stadium attendance. Supporters seem to prefer to watch football on TV rather than watch matches live at a stadium (see Abdul-Hamid and Kendall (2008) for more information).

The objective of this study is to model the problem in a way that can increase stadium attendance by scheduling important matches in a timeslot such that it motivates supporters to attend the game. In order to achieve this, we propose to model the problem so that we can solve it using mathematical programming.

The rest of this abstract is structured as follows. Section 2 discusses related work. Section 3 provides more details about the problem, and section 4 presents our mathematical model. We summarise our work and highlight future work in section 5.

2 Related Work

There are many papers that address sports scheduling problems from the real world. For example, Bartsch et al. (2006) have scheduled professional soccer leagues of Austria and Germany. Models and algorithms are developed which generate schedules and have been adopted by the Deutsche Fussball-Bund (DFB) and Österreichische Fussball-Bund (OFB). Duran et al. (2007) tackle the problem of Chileans' interest in football by modeling the problem using integer linear programming. Kendall (2008) schedules English fixtures over holiday periods in order to minimize travel distances.

Briskorn and Drexl (2007) consider the more general issue of round robin tournament scheduling using Integer Programming (IP), and utilise CPLEX to find optimal solutions for small-to-medium-size instances. Lee et al. (2006) consider the Travelling Tournament Problem, formulating it as an IP and use Tabu Search to reach the optimal solution.

The focus of this work is on maximising gate attendance. A recent work by Buraimo et al. (2009) model match attendance, yielding results beneficial for the English football clubs to take decisions such as when to schedule midweek games. Maximising gate attendance is often associated with how important or exciting a match is. Scarf and Shi (2008) work on the importance of a match in a tournament; using logistic regression to predict matches and Monte Carlo simulation to compute the match importance measure, which they apply to soccer matches in the English Football Association Premier League. Vecer et al. (2007) illustrate the concept of excitement at soccer games using Poisson model of scoring. They relate the excitement to the variability of the win expectancy; the larger the variability, the higher the excitement.

Nor Hayati Abdul Hamid
University of Nottingham
E-mail: nha@cs.nott.ac.uk

Graham Kendall
University of Nottingham
E-mail: gmk@cs.nott.ac.uk

Mujgan Sagir
Eskisehir Osmangazi University
Email: mujgano@ogu.edu.tr

3 Problem Definition

The Malaysian Super League (MSL) is structured as a double round robin tournament, such that each team has to play against each other once at home and once away. For the MSL, six matches are played in a given time slot giving 26 time slots for the whole season. Thirteen time slots are defined as the first round, and the next 13 for the mirrored, second round. MSL matches are played on weekdays (Tuesday or Wednesday) and at weekends (Saturday or Sunday). Ideally, if a team plays home, it would play away on the next game. However, sometimes there are occasions when a team has to play consecutive home or away games. It is also observed that if a team plays at home on the first day of the season, they play away on the last day of the season (and vice versa). The current schedules are produced manually by the Football Association of Malaysia (FAM).

In our work, we plan to produce feasible schedules but also maximise the assignment of important matches to weekdays while maximising the assignment of unimportant matches to weekends. The rationale is that if a match is important, whether it is played on weekdays or weekends, it will attract supporters to the stadium. By allocating important matches during weekdays, the weekend slots can be assigned to unimportant matches on the assumption that supporters have free time to watch matches at stadium.

The following constraints will be considered during the modeling:

- There will be six fixtures per time slot.
- Each fixture should be assigned to a timeslot.
- A team cannot have more than two consecutive home games.
- A team cannot have more than two consecutive away games.
- Each team is scheduled 24 time slots and also scheduled a *bye* in each round.
- If a team plays home during the first round, it must play away during the second round.

4 Mathematical Model

The integer linear programming model used to generate the fixture schedules is presented below. The following notations are defined for this model.

Indices and sets

$N = 13$ (Number of teams)

$T = 26$ (Number of timeslots)

$Q = 156$ (Number of fixtures)

$n = 1 \dots N$ (index for teams)

$t = 1 \dots T$ (index for timeslots)

$q = 1 \dots Q$ (index for fixtures)

C is a set of teams, $c_n \in C$

F is a set of fixtures, $f_q \in F$

T is a set of timeslots, $t \in T$

T_1 is the set of weekday timeslots, $T_1 \subset T$

T_2 is the set of weekend timeslots, $T_2 \subset T$

$T = T_1 \cup T_2$

Q_1 is the index of important fixtures

Q_2 is the index of unimportant fixtures

L_q is the level of importance for fixture q (the parameters are obtained from the AHP analysis)

Each fixture, f_q , is a tuple = $\{c_k(\text{home team}), c_m(\text{away team})\}$

$c_k \subset C, c_m \subset C, k \neq m$ (tuples are prepared earlier)

$$f_{qj} = \begin{cases} 1, & \text{if team } j \text{ is in fixture } q \\ 0, & \text{otherwise} \end{cases}$$

$$\alpha_{nq} = \begin{cases} 1, & \text{if team } c_n \text{ is playing at home in fixture } f_q \\ 0, & \text{otherwise} \end{cases}$$

Decision Variables

$$x_{qt} = \begin{cases} 1, & \text{if fixture } f_q \text{ is scheduled in timeslot } t \\ 0, & \text{otherwise} \end{cases}$$

Objective function

Maximise

$$f = \sum_{q \in Q_1} \sum_{t \in T_1} L_q x_{qt} - \sum_{q \in Q_2} \sum_{t \in T_2} L_q x_{qt} \quad (1)$$

Subject to

For each t , the total number of scheduled matches must be six.

$$\sum_{q=1}^Q x_{qt} = 6 \quad \forall t \quad (2)$$

Each fixture q should be assigned to a time slot.

$$\sum_{t=1}^{26} x_{qt} = 1 \quad \forall q \quad (3)$$

A team cannot have more than two consecutive home games.

$$\sum_q x_{qt} \alpha_{nq} + \sum_q x_{q(t+1)} \alpha_{nq} + \sum_q x_{q(t+2)} \alpha_{nq} \leq 2 \quad \forall (n,t), (t = 1, \dots, 11) \quad (4)$$

$$\sum_q x_{qt} \alpha_{nq} + \sum_q x_{q(t+1)} \alpha_{nq} + \sum_q x_{q(t+2)} \alpha_{nq} \leq 2 \quad \forall (n,t), (t = 14, \dots, 24) \quad (5)$$

A team cannot have more than two consecutive away games.

$$\sum_q x_{qt} (1-\alpha_{nq}) + \sum_q x_{q(t+1)} (1-\alpha_{nq}) + \sum_q x_{q(t+2)} (1-\alpha_{nq}) \leq 2 \quad \forall (n,t), (t = 1, \dots, 11) \quad (6)$$

$$\sum_q x_{qt} (1-\alpha_{nq}) + \sum_q x_{q(t+1)} (1-\alpha_{nq}) + \sum_q x_{q(t+2)} (1-\alpha_{nq}) \leq 2 \quad \forall (n,t), (t = 14, \dots, 24) \quad (7)$$

For the consecutiveness constraints, in (4) and (5), we take the summation over q for specific timeslots t , $t+1$ and $t+2$ and the number of assignment for a specific team n , should be less than or equal to two. We check the existence of a specific team n in each fixture with the variable α_{nq} . The constraint does not allow any team which is in all three fixtures to be assigned in three consecutive periods as home team.

Same applies for (6) and (7) for the reverse fixture.

For constraint (8), each team is going to be scheduled at 24 of 26 time slots. Each team is going to be scheduled as *bye* for one time slot in both rounds.

$$\sum_{t=1}^T x_{qt} \alpha_{nq} + x_{qt} (1-\alpha_{nq}) = 24 \quad \forall n \quad (8)$$

For a t , a fixture is going to be scheduled either when team n is an away team or home team. So for each t , team n is going to play as either one. In 26 time slots in total, this will happen 24 times. For the two time slots, there will not be any assignment for a team. In order to extract the team information from the fixture definition, we relate the decision variable with α_{nq} .

For constraint (9), each team has to play once against the other, for the first round.

$$\sum_{t=1}^{T/2} x_{qt} f_{qh} f_{qz} = 1 \quad \forall (h,z) \quad (9)$$

For constraint (10), each team has to play once against the other, for the second round.

$$\sum_{t=(T/2)+1}^T x_{qt} f_{qh} f_{qz} = 1 \quad \forall (h,z) \quad (10)$$

For constraint (11), if a team plays against another at home this round, it should be away on the second round.

$$\sum_{t=1}^{T/2} x_{qt} \alpha_{nq} f_{qn} f_{qz} = 1 - \sum_{t=(T/2)+1}^T x_{qt} \alpha_{nq} f_{qn} f_{qz} \quad \forall (h,z) \quad h \neq z \quad (11)$$

Dimensional analysis

To gauge the size of the model, we calculate the total number of constraints and variables which correspond to our real data set. Table 1 presents this analysis.

Table 1 Problem size

Constraint no.	Indices	Total number of constraints	For the problem
2	T	T	26
3	Q	Q	156
4	t, n	$((T/2)-2)*n$	143
5	t, n	$((T/2)-2)*n$	143
6	t, n	$((T/2)-2)*n$	143
7	t, n	$((T/2)-2)*n$	143
4,5,6,7 apply when T is an even number			
8	n	n	13
9	h, z	$\binom{N}{2}$	78
10	h, z	$\binom{N}{2}$	78
11	h, z	$\binom{N}{2}$	78
Variable	Indices	Total number of variables	
x_{qt}	q, t	$Q*T$	
Total number of constraints		$T+Q+ 4(((T/2)-2)*n)+3*\binom{N}{2}$	1,157
Total number of variables		4,056	

In total the model has 1,157 constraints which should be tractable to solve by exact methods.

5 Conclusion and future work

The decrease in gate receipts in Malaysian football is perceived as a problem. We propose to address this by modeling the problem with the objective of maximising the number of supporters by assigning important games to weekdays while assigning the unimportant ones to weekends. It is based on the assumption that if the match is important, supporters will go to stadium to watch it.

We now plan to use mathematical programming to find the optimal solution to the problem presented above. If an exact solution cannot be found, we will resort to heuristics to find good quality solutions.

We will report our findings during our conference presentation.

References

1. Abdul-Hamid, N. and Kendall, G., *Maximising Stadium Attendance: A Case Study of Malaysian Football*, In: Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling PATAT, (E. Burke. and M. Gendreau., Eds.), 1-4, Montreal, Canada, (2008).
2. Bartsch, T., Drexl, A., Kröger, S., *Scheduling the professional soccer leagues of Austria and Germany*, Computers and Operations Research, 33, 1907-1937, (2006).
3. Briskorn, D., Drexl, A. *IP models for round robin tournaments*, Computers & Operations Research, 36, 837-852, (2009).
4. Buraimo, B., Forrest, D, Simmons, R., *Insights for clubs from modeling match attendance in football*, Journal of Operational Research Society, 60(2), 147-155, (2009).
5. Duran, G., M. Guajardo, et al., *Scheduling the Chilean soccer league by Integer Programming*. Interfaces, 37(6): 539-552. (2007).
6. Kendall G., *Scheduling English Football Fixtures Over Holiday Periods*. Journal of the Operational Research Society, 59(6), 743-755. (2008).
7. Lee et al., *Mathematical Modeling and Tabu Search Heuristic for the Traveling Tournament Problem*. In: M. Gavrilova et al. (Eds) ICCSA 2006, LNCS 3982. Berlin Heidelberg: Springer-Verlag, 875-884, (2006).
8. Scarf P., Shi X., *The importance of a match in a tournament*. Computers and Operations Research, 35(7): 2406-2418, (2008).
9. Vecer, J., Ichiba, T., and Laudanovic, M., *On Probabilistic Excitement of Sports Games*, Journal of Quantitative Analysis in Sports: Vol. 3 : Iss. 3, Article 6, (2007).

Soccer schedules in Europe: an overview

Dries R. Goossens • Frits C.R. Spijksma

1 Introduction

Most national soccer championships in Europe consist for the largest part (if not fully) of a round robin tournament. There are however differences in the number of teams taking part, and in the number of times each team meets each other team. In most countries, the double round robin tournament is used, although triple (e.g. Denmark, Albania, Croatia) and quadruple (e.g. Austria, Armenia) round robin tournaments exist as well. As far as we are aware, no single round robin tournaments are used for national soccer championships in Europe. In some countries (e.g. The Netherlands, Malta), the regular competition is followed by a play-off stage, which decides on the league championship, qualification for European competitions, and/or relegation.

In section 2, we give an overview of a number of characteristics of schedules for national soccer championships in Europe for the seasons 2002-2003 till 2007-2008. In particular, we investigate to what extent the so-called canonical schedule is still used in soccer scheduling. We also look into the number of breaks and the maximal length of a break that have been tolerated in the schedules. Further, we determine the popularity of various schemes to schedule the second (or subsequent) part(s) of the tournament based on the first. Finally, we examine for each schedule the balancedness of carry-over effects and the spread of the top games. In section 3, we compute the importance of each fixture in each of Europe's most important soccer leagues. This gives us an idea which competition is the most attractive with respect to suspense and tension.

Dries R. Goossens
PostDoc researcher for Research Foundation – Flanders
Center for Operations Research and Business Statistics (ORSTAT)
Faculty of Business and Economics - Katholieke Universiteit Leuven
Naamsestraat 69, 3000 Leuven, Belgium
E-mail: Dries.Goossens@econ.kuleuven.be

Frits C.R. Spijksma
Center for Operations Research and Business Statistics (ORSTAT)
Faculty of Business and Economics - Katholieke Universiteit Leuven
Naamsestraat 69, 3000 Leuven, Belgium
E-mail: Frits.Spijksma@econ.kuleuven.be

2 Schedule characteristics

Given a round robin competition with an even number of n teams, a schedule can be seen as a partitioning of the edges of a complete graph with n nodes (K_n) into one-factors (i.e. perfect matchings). For each round in the competition, there is a one-factor showing which teams play against each other. There is a specific partition of the edges of K_n into one-factors, that is called the canonical 1-factorization. With the canonical 1-factorization, the 1-factor F_t ($t=1, \dots, n/2-1$) contains the edge $[n-1, t-1]$ and all edges $[(t+k) \bmod (n-1), (t-k) \bmod (n-1)]$ for $k=1, \dots, n/2-1$ (notice that the nodes are labeled $0, 1, \dots, n-1$). Griggs and Rosa (1996) found that for the season 1994-1995, 16 out of the 23 schedules they examined were based on canonical 1-factorizations. We investigate whether the popularity of the canonical schedule still holds a decade later, when mathematical programming has found its way into the practice of scheduling soccer competitions (see e.g. Bartsch et al. (2006) and Goossens and Spieksma (2009)).

A succession of home (away) games is called a break. In many competitions, it is imperative that breaks have a length of at most two, and that the total number of breaks in the schedule is minimal. Nevertheless, in some countries, breaks of length three (or more) are allowed (e.g. Italy), or an equal distribution of the breaks over the teams is perceived more important than a minimal number of breaks for the competition as a whole. We give an overview of the occurrence of breaks in various soccer schedules.

In case of a double round robin tournament, the second half of the competition can usually be derived from the first. In most countries (e.g. Germany, Spain), it is considered fair to schedule the second half of the competition identically to the first, but with the home advantage inverted. This is called mirroring. Another possibility is the so-called English scheme, where games in the first and the last round are identical, as are fixtures from round $n-1+t$ and round $t+1$ with $t=1, 2, \dots, n/2-1$ (again with the home advantage inverted). This scheme is played for instance in France. In some countries (e.g. England, The Netherlands), a separation constraint is used when creating the schedule, enforcing that there should be at least k rounds between two games with the same opponents, which leaves more possibilities to fine-tune the schedule to the wishes of the stakeholders than compared to the mirroring or the English scheme. For triple or quadruple round robin tournaments, variants of these schemes can be applied.

In order to enforce that for each team, tough and weak opponents are distributed evenly over the season, the teams can be partitioned in so-called strength groups. The schedule can then be such that no two teams belonging to the same strength group are scheduled against each other in two consecutive rounds (see also Briskorn, 2008). In some countries (e.g. The Netherlands), it is a goal to have all top games on the same day (a so-called “Super Sunday”), whereas other countries strive to spread these games as much as possible throughout the season (e.g. Belgium). Furthermore, for many teams, home games against top teams are financially quite important, insofar that teams have a disadvantage when confronted with a schedule that leaves them without such home games for half a season. We find out for each competition what policy with respect to spreading the top games results from their schedules.

In a round robin tournament, we say that a team i gives a carry-over effect to a team j , if some other team t 's game against i is immediately followed by a game against team j . It is often assumed that these effects can have an influence on the outcome of the tournament. Indeed, if team i is a very strong team, then its opponent, team t , could be exhausted and discouraged after this game, which could benefit its next opponent, team j . Clearly, carry-over effects are unavoidable in a schedule. Nevertheless, a fair schedule would be one where carry-over effects are balanced as much as possible. We investigate to what extent soccer schedules in Europe manage to balance these carry-over effects.

3 Attractiveness of the competition

The second part of our overview deals with the importance of matches. We consider a match to be unimportant with respect to some outcome (e.g. relegation, championship), if it cannot have an influence on that outcome for any of the competing teams. Bojke (2007) confirms that games in the English Premier League between two teams that have nothing to gain or to loose attract less fans to the stadium.

We define the importance $I_m(X)$ of a match m with respect to outcome X as

$$I_m(X) \begin{cases} = 2n_m(X)/A_m(X)^2 & \text{if } A_m(X) > 0 \\ = 0 & \text{if } A_m(X) = 0 \end{cases}$$

where $A_m(X)$ is the number of teams that are still competing for outcome X at the time when match m is played, and $n_m(X)$ is the number of teams in match m that are still competing for outcome X . We scaled this measure with a factor 2, to make sure that the importance of a match coincides with the interval $[0,1]$. With our measure, the importance of a match with respect to some outcome increases when less teams are in the running for that outcome (provided of course that there are still teams competing for that outcome). Indeed, when only 2 teams are in the running for e.g. the league championship, their games will be highly important. If one team loses its game, it could be enough that its competitor for the league championship wins to decide the championship. Contrarily, when a team is still in the race for the championship together with 5 other teams, the fact whether or not this team will be league champion will depend on the results of many other games, thus reducing the importance of this team's game. We refer to Goossens et al. (2008) for a more detailed discussion of this measure. Using the actual results of the seasons 2002-2003 till 2007-2008, we compute for each match in each of the major competitions its importance with respect to the league championship, qualification for European football, and relegation. This gives us an impression of the attractiveness of these competitions.

References

1. Bartsch, T., Drexler, A., Kroger, S., Scheduling the professional soccer leagues of Austria and Germany. *Computers and Operations Research*, 33, 1907–1937 (2006)
2. Bojke C., The impact of post-season play-off systems on the attendance at regular season games. In: Albert, J., Koning, R.H. (eds.) *Statistical Thinking in Sports*, pp 179-202 (2007)
3. Briskorn, D., Sport leagues scheduling: models, combinatorial properties, and optimization algorithms. *Lecture Notes in Economics and Mathematical Systems*, Vol. 603, Springer, Berlin (2008)
4. De Werra, D., Geography, games, and graphs. *Discrete Applied Mathematics* 2, 327-337 (1980)
5. Goossens, D., Spieksma, F., Scheduling the Belgian soccer league. *Interfaces*, forthcoming (2009).
6. Goossens, D., Beliën, J., Spieksma, F., Comparing league formats with respect to match importance in Belgian football. *FBE Research Report KBI_0830*, 1-21 pp. K.U.Leuven (2008)
7. Griggs, T.S., Rosa, A., A tour of European soccer schedules, or testing the popularity of GK_{2n} . *Bulletin of the Institute of Combinatorics and its Applications* 18, 65-68 (1996)
8. Rasmussen, R.V., Trick, M.A., Round robin scheduling - a survey. *European Journal of Operational Research* 188, 617-636 (2008)

Round-robin Sports Scheduling from a Graph Colouring Perspective

A Case Study in Rugby Union Scheduling

Rhyd Lewis · Jonathan Thompson.

1 Introduction

In this study we examine the problem of producing sports leagues that are “round-robin” in structure. Such structures occur when we have n teams (or competitors) in a league, and each team is required to play all other teams exactly m times within a fixed number of rounds. Examples include the Six Nations Rugby Championship (where $n = 6$ and $m = 1$), the England and Wales County Cricket Championship ($n = 9$, $m = 2$), and most European and South American national soccer leagues. In practice the most common league structures are *single* and *double* round-robins, where $m = 1$ and 2 respectively.

It has been known for many years [8] that valid round-robin schedules using the minimal $m(n-1)$ rounds can be constructed for any positive integer n . It is also known that the number of *distinct* round-robin schedules grows exponentially with n , since this figure is monotonically related to the number of non-isomorphic one-factorisations of a complete graph with n vertices (which is also known to increase exponentially with n). Such a growth rate implies that, for non-trivial problems, examining all possible schedules in order to identify the one that best fits the needs of the user will not always be possible in reasonable time (in practice, “non-trivial” values for n would appear to be anything above 10 when $m = 1$ [4].)

Recent work on round-robin scheduling problems has tended to focus on the use of metaheuristic-based approaches, which have shown to be effective for exploring the large search spaces associated with such problems [1,3,10,12]. The strategy used in such methods is to apply a two-stage scheme, whereby a valid round-robin schedule is first constructed, with changes then being made to the schedule via the application of neighbourhood operators that preserve this validity. Until now, such approaches have tended to focus on a specific formulation of round-robin scheduling problem, known as the Travelling Tournament Problem, originally proposed by Easton [5].

In this work we have chosen to look at the problem of round-robin scheduling in more general terms. Specifically, we choose to view it as a type of graph colouring prob-

Cardiff School of Mathematics,
Cardiff University, CF24 4AG,
Wales.
E-mail: LewisR9/ThompsonJM1@cf.ac.uk

lem. Graph colouring is a classical combinatorial optimisation problem for which much research has been conducted, both from the perspective of *solving* graph colouring problems [2,6,7,9] and in *exploring* the spaces of feasible and/or optimally coloured graphs [9,11,13]. By using such principles, in this work we analyse the ability of a number of well-known graph colouring algorithms for solving round-robin scheduling problems of different sizes. We also show how the graph colouring model can be extended in order to cope with additional hard (i.e. mandatory) constraints that can often be imposed in practical situations. Finally, we also examine a number of neighbourhood operators, originating from the graph colouring literature, that can then be utilised in order to explore the space of feasible round-robin schedules.

In the final part of this work we consider a complex real world scheduling problem that was given to us by the Welsh Rugby Union, based at the Millennium Stadium in Cardiff, Wales. This problem involves producing a double round-robin tournament using $2(n - 1)$ rounds and features a number of additional hard constraints to do with stadium sharing, stadium availability, and pre-assignment issues. In addition, soft constraints involving the spread of related matches throughout the tournament are also imposed. We propose two separate algorithms for tackling this problem, both that make use of our proposed algorithmic operators, and we demonstrate their ability to produce solutions that are superior to those currently used by the league, in short periods of time.

References

1. Anagnostopoulos, A., Michel, L., van Hentenryck, P., Vergados, Y.: A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling* **9**(2), 177–193 (2006)
2. Culberson, J., Luo, F.: Exploring the k-colorable landscape with iterated greedy. In: D.S. Johnson, M.A. Trick (eds.) *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, vol. 26, pp. 245–284. American Mathematical Society (1996)
3. Di Gaspero, L., Schaerf, A.: A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics* **13**(2), 189–207 (2007)
4. Dinitz, J., Garnick, D., McKay, B.: There are 526,915,620 nonisomorphic one-factorizations of K_{12} . *Journal of Combinatorial Designs* **2**, 273–285 (1994)
5. Easton, K., Nemhauser, G., Trick, M.: The traveling tournament problem: description and benchmarks. In: T. Walsh (ed.) *Principles and Practice of Constraint Programming, Lecture Notes in Computer Science*, vol. 2239, pp. 580–585. Springer, Berlin/Heidelberg (2001)
6. Galinier, P., J.-K., H.: Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization* **3**, 379–397 (1999)
7. Jensen, T.R., Toft, B.: *Graph Coloring Problems*, 1 edn. Wiley-Interscience (1994)
8. Kirkman, T.: On a problem in combinations. *Cambridge Dublin Math Journal* **2**, 191–204 (1847)
9. Lewis, R.: A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing. *Computers and Operations Research* **36**(7), 2295–2310 (2009)
10. Lim, A., Rodrigues, B., Zhang, X.: A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operational Research* **174**(3), 1459–1478 (2006)
11. Morgenstern, C.: *Algorithms for general graph coloring*. Ph.D. thesis, University of New Mexico (1989)
12. Ribeiro, C.C., Urrutia, S.: Heuristics for the mirrored travelling tournament problem. *European Journal of Operational Research* **179**(3), 775–787 (2007)
13. Thompson, J., Dowsland, K.: A robust simulated annealing based examination timetabling system. *Computers and Operations Research* **25**(7/8), 637–648 (1998)

Workforce Planning and Scheduling for the HP IT Services Business

Cipriano A. Santos, Alex Zhang, Maria Teresa Gonzalez, Shelen Jain, and Andrew Bye

1 Introduction

Our research aims at developing analytical models for workforce planning to support labor supply and demand matching in an uncertain environment while minimizing costs of service delivery. The research challenges lie in (a) developing a modeling framework that encompasses the vast diversity of the IT services business from project-based to annuity-based and many other variations in between; (b) forecasting the labor demand for thousands of skills but with sparse historical data and hidden structural changes in the data; (c) building tractable optimization models for demand and supply matching – when both are uncertain; (d) aligning employees’ career paths with business needs; and lastly (e) addressing the sheer scale of the problem due to multiple geographies, hundreds of service offerings, thousands of skills and tens of thousands of people resources. Our focus in this paper is on workforce planning and scheduling at a single project-based service delivery center.

Workforce planning and scheduling has been studied in the Operations Research literature under the name of Manpower (Personnel) Planning since 1950s. In fact, one of the early applications of the linear programming method pioneered by George B. Dantzig is “program planning of interdependent activities” (Dantzig [1]). There are many papers particularly for the U.S. Armed Forces on Manpower Planning. Saul I. Gass, a leading researcher in the field, defines Manpower Planning as the problem to determine the number of personnel and their skills that best meet the future operational requirements of an enterprise, and proposes that Manpower Planning should be tackled by a hierarchical planning approach (Gass [2]). Gass proposes to use Markov chains and network flow models to explain and optimize workforce dynamics. Grinold and Marshall [4] give the mathematical treatment of Markov chains and network flow models for Manpower Planning. Few business organizations face the type of workforce planning and scheduling problem at HP Services in terms of labor skills variety, complexity, supply dynamics, demand uncertainty, and competitive marketplace. The work by Gresh et al [3] done at IBM TJ Watson Research Center for IBM Global Services is most relevant to our workforce planning and scheduling setting. There are a number of aspects which the IBM work does not address. IBM’s tool does not assign work to individuals but to a group of people matching skills and attributes. People substitution (as a group) to

Cipriano A. Santos
HP Laboratories
E-mail: cipriano.santos@hp.com

Alex Zhang
HP Laboratories
E-mail: alex.zhang@hp.com

allow flexible skill matching is part of the IBM tool. However, each employee is assumed to have only one skill. Training of employees so they can handle different jobs depending on the demand mix is not addressed; nor is future hiring allowed. Hiring plans are followed as part of the input, rather than optimized together with the master plan. Another aspect is that IBM tool does not model personnel attrition. Finally, demand and supply in the IBM tool are treated as deterministic.

2 Workforce planning and scheduling for project-based services

Workforce planning and scheduling entails anticipating supply availability and job requirements in order to have the right people, with the right skill, at the right time, in the right place, at the right cost. The problem of matching future availability of employees with future job requirements is quite complex since HP has hundred of thousands of employees in project delivery roles, with thousands of skills distributed in countries all over the world. The other challenge of matching future availability of employees with future job requirements is that supply and demand is uncertain. Finally, there are no clear costs when matching an employee to a job. For example, let's suppose we require a Java programmer at the Expert level for the manufacturing industry (job *j*), and have an employee (*e1*) that is a Java programmer at the Intermediate level from the Aerospace Industry, and an employee (*e2*) that is a Java programmer at the Master level from the Telecom Industry, then it is not clear what is the cost of assigning *e1* to *j* and *e2* to *j*. There is the problem of discriminating *good* matches from *bad* matches automatically when matching future availability of employees with future job requirements. In practice, first-line managers *know* their employees and can assign the right employee to the right job, a *good* match; however at a company like HP where we have hundred of thousands of employees and jobs we will like to formulate a business utility function that automates the discrimination of good matches from bad matches.

A typical Project Delivery Center (PDC) faces both demand and supply uncertainties in workforce management. A PDC has a pipeline of potential projects called the *funnel of opportunities*. The main sources of demand uncertainty are the uncertainty about winning a project opportunity and the uncertainty about when the project will start if won. The types and amounts of resources required by the project are also uncertain at the time of resource planning for the projects. On the other hand, the main sources of supply uncertainty are the uncertainty about workforce attrition and uncertainties around hiring. Hiring lead times can vary from a couple of weeks to several months; therefore we need to plan ahead of demand realization in order to make sure that demand will be satisfied. Therefore the main objective of workforce planning and scheduling is to balance workforce utilization and availability to optimize demand fulfillment. We tackle the workforce planning and scheduling problem in two phases. Phase 1 is a demand and supply consolidation step to express demand and supply in terms of jobs. At Phase 2 we formulate a large scale mixed integer programming model to optimally match future available employees with future job requirements.

2.1 Supply and Demand Consolidation

During our initial research we discovered that workforce planning under uncertainty cannot be done at the detailed skill-set level; instead, we need to plan at a higher level of aggregation than detailed skill set of requirements. We call this high level skill *Skill-Group* (SG). Skill Groups allow mapping the demand (the Funnel of project opportunities) into the language of the supply (employee skills). The key ideas to determine the Skill Groups is to first identify standard projects of a given IT technology. Then identify the standard jobs that each standard project requires. For each a standard job, we identify the main technical skills required, this set of skills constitutes a SG, i.e. the main technical skills required to perform a standard job defines the SG. The PDC identified five more job attributes required to properly allocate people to projects; therefore the final labor *nomenclature* considered includes the

following attributes: Skill Group – J2EE, dot net, ABAP, etc; Workforce Type – regular or contingent; Job Level – entry, intermediate, expert, master, etc; Business Domain – manufacturing, aerospace, finance, etc; Location Offshore –Bangalore, Chennai, etc; Location On site – USA, Germany, UK, etc.

The main function of the Supply and Demand Consolidation approach is to map job requirements of project opportunities in the Funnel into employees qualified to perform the job required, where a job is represented as a string of job attributes. Our approximate matching approach allows mapping employees to jobs at less than 100% match and provides a score for the matching. To tackle demand and supply uncertainty, we create a buffer capacity called *bench*. To explain how our approach builds a bench, we first need to describe how we parameterize demand and supply uncertainty. Each opportunity in the funnel has a win probability that measures the degree of confidence we have in winning such a project, a (forecasted) schedule of headcount requirements for each job required to deliver the project, and 3 points in time where the project can start – Earliest Starting Time, Expected Starting Time, and Latest Starting Time. To characterize supply uncertainty, we consider attrition rates of employees at the various technologies of the PDC. We observe that winning projects in the Funnel are independent events, thus the headcount requirements for a job of a project defines a Bernoulli process. We assume that the attrition process can be modeled with a binomial distribution with probability of success equal to the attrition rate and the population of the distribution equal to the inventory of employees; this binomial distribution determines the number of employees that need to be replaced. Resource Managers (RM) at the PDC define a threshold of win probability such that opportunities with a win probability higher than or equal to the threshold will get employee resources to satisfy the headcount requirements schedule for each job required. For opportunities with a win probability less than the threshold and for resources that need to be replaced due to attrition, we compute the SG requirements cumulative distribution as the convolution of the Bernoulli and Binomial distributions. We interpret the threshold defined by the Resource Managers as the Service Level (SL) required at the PDC, then based on the SL we determine the number of headcount requirements for SG that we need to have as a buffer capacity – i.e. bench.

The main outputs of the Demand and Supply Consolidation phase are a *qualifications* table that determines which jobs required by the opportunities in the Funnel and Bench an employee is 100% qualified for, and a *transitions* table that determines which jobs required by either opportunities or Bench an employee is partially qualified for. The transitions table defines the score of the employee-job matching and the type of workforce transformation (training, promotion/demotion, re-location) required to fully qualify the employee for the job.

2.2 Mixed integer programming model to determine a workforce plan and schedule

We have built a mixed integer programming model that balances the trade-off between workforce utilization and availability to optimize demand fulfillment. The MIP tool generates three plans: a fulfillment plan, an employee transformation plan, and a hiring plan. The fulfillment plan encompasses project-job-employee-period allocations and a planned *bench*. A particular use of bench is called *practice building* where new employees will be hired to build expertise in a new technology field. The employee transformation plan entails employees' training, promotions, and re-deployment; i.e. anything that changes the current status of an employee to satisfy job requirements. The employee transformation plan identifies opportunities to increase resource utilization and aligns employees' career paths with business need of satisfying project requirements. Remaining gaps that cannot be filled by the fulfillment plan and employee transformation plan are filled by hiring new employees considering hiring lead times. The pipeline of joiners (new hires) is considered part of the hiring plan. Remaining gaps need to be resolved manually by Practice Managers (PM) and RM. Considering the SG requirements cumulative distribution computed at the Demand and Supply Consolidation phase, we can determine the probability of satisfying SG requirements with the workforce plans generated by the MIP (fulfillment, transformation, and hiring plans). This *service level*

metric is particularly valuable when management imposes a hire freeze or limits the number of people to hire, allowing PM to negotiate hiring for critical SG requirements. In addition to the mathematical optimization logic, the MIP encodes business rules that reflect the preferences RM and PMs in allocating employee resources. For example, high-priority opportunities in the funnel (demands) are staffed before low priority ones; low priority opportunities are to be filled with pipeline joiners (new hires). Employees who have been “on the bench” for a longer duration will have a higher priority to be assigned to projects.

3 Conclusions and future research

The workforce planning and scheduling approach presented has been deployed at an HP Project Delivery Center (PDC). Our approach uses commercially available solvers to tackle the mixed integer programming model. The typical problem is very large; however, we have observed that these MIP solvers reduce the original model size to less than 10% with their constraint programming algorithms used during pre-processing. In addition, we have observed that the underlying MIP formulation is tight, since most of the time the optimal solution of the LP relaxation of the reduced problem is integer-feasible. Therefore, we believe that there is an underlying network flow structure that can be exploited. As future research, we want to study the *Polyhedron* associated with the MIP formulation and derive efficient algorithms to solve the problem. Also, we are building a simulation tool that will help to evaluate directly the effects of demand and supply uncertainty in terms of metrics such as demand fulfillment and resources utilization.

Acknowledgements We will like to thank Mr. Babu Rangaswamy and his team. Mr. Ranga Ramanujam, Mr. Ravindra Kumar, and Mr. Pradeep Rao for their leadership in taking our ideas about workforce planning and scheduling and further having implemented the planning tool at the Global Delivery Applications Center in Bangalore India.

References

1. Dantzig, G.B. Programming of Interdependent Activities, II: Mathematical Model, *Econometrica*, Vol.17, 200-211 (1951)
2. Gass, Saul I. Military manpower planning models, *Computers & Operations Research*, 18 (1), 65-73 (1991)
3. Gresh, D.L., D.P. Connors, J.P. Fasano, R.J. Wittrock. Applying Supply Chain Optimization Techniques to Workforce Planning Problems. *IBM Journal of Research & Development*, Vol.51 (2007), No.3/4, May/July 2007, pp.251-261 (2007)
4. Grinold, R.C. and K.T. Marshal. *Manpower Planning Models*, North-Holland, Amsterdam (1977)

Integrated workforce staffing and scheduling

A case study at a major aircraft maintenance company

Jeroen Beliën · Erik Demeulemeester

1 Introduction

Labor costs are amongst the most important costs for companies in highly developed countries. While manufacturers move their production units en masse to low wage countries, service companies are often tied to their current locations. Unlike production, the service surplus value is often directly related to the place where the service is provided. Moreover, service companies generally require more highly educated employees who tend to be scarce in low wage countries. Therefore, modern staffing and rostering software can have great value for these kinds of service companies. The methodology described in this paper was successfully applied for the employee staffing and scheduling problem of Sabena Technics, a major aircraft maintenance company. Sabena Technics takes care of the maintenance for over 300 flights, arriving at and departing from Brussels Airport in a fixed weekly schedule. The maintenance of aircraft needs to be done by highly qualified employees and cannot simply be moved to a low wage country airport.

2 Model

Recently, staff scheduling and rostering problems have been subject to many research efforts (see, e.g., [4] for a general literature overview or [3] for a more recent overview on nurse rostering). More specifically, [1] presents a case study on an aircraft maintenance workforce scheduling problem. This paper presents a mixed integer optimization model for a workforce staffing and scheduling problem encountered at Sabena Technics. Work is organized in teams and the team rosters follow a cyclic pattern. This

Jeroen Beliën

Center for Modeling and Simulation, Hogeschool Universiteit Brussel, affiliated researcher Center for Operations Management, Faculty of Business and Economics, Katholieke Universiteit Leuven

E-mail: jeroen.belien@hubrussel.be

Erik Demeulemeester

Center for Operations Management, Faculty of Business and Economics, Katholieke Universiteit Leuven

E-mail: erik.demeulemeester@econ.kuleuven.be

workforce staffing and scheduling problem has several interesting characteristics. First, the workload (expressed in man-hours) in the coverage constraints is not a fixed value. Each aircraft maintenance needs to take place within a given time window (bounded by the scheduled time of arrival and the scheduled time of departure), but there is certain flexibility with respect to the exact timing of the maintenance and the number of persons working on a particular aircraft. Hence, the timing of the workload is an extra decision in the model. Second, in the literature, staffing and scheduling are almost always considered as two separate problems (see e.g. [5], [6], [7]). In a first phase the staffing problem is solved, which results in workforce sizes. Next, these workforce sizes are input for the scheduling problem. In practice, however, the staffing and scheduling problem is sometimes solved at the same moment. This entails interesting opportunities, as the solution to the staffing problem has a serious impact on the scheduling problem. A model that integrates the staffing and the scheduling problem could take advantage of this dependency. The integrated staffing (i.e., determining how many employees should be hired) and scheduling (i.e., determining when these employees should work) is basically a non-linear problem. We propose an enumeration approach in which a mixed integer linear problem is solved for every interesting combination of team sizes and weeks in the roster cycle. This leads to several optimized rosters of which the company can choose the one they like most. Third, since flight arrivals and maintenance workloads are subject to uncertainty, the workforce staffing and scheduling model is extended with extra constraints increasing the robustness of the generated rosters. [2] presents an alternative approach, based on simulation, to cope with uncertainty. Fourth, we show how part-time workers can be included in the generated rosters.

3 Results

The model was used to examine different scenarios (introduction of a special weekend cycle, relaxation of the weekend constraints, different capacity buffer sizes and time buffer sizes, etc.) and to generate the new work rosters for the 2008 winter season. For each scenario the algorithm returned over 50 feasible solutions most of them within 5% of the theoretic optimum (current lower bound). The most interesting solutions (in terms of costs) were further analyzed by the company and were subject to an intensive bargaining process. Using the feedback of these bargaining processes the model was adapted in order to achieve an acceptable workforce schedule. Generating new rosters after each bargaining process required several hours (days) of computation time. After 7 bargaining rounds, the finally selected schedule resulted in a saving of 6%.

References

1. Alfares, H.K., Aircraft maintenance workforce scheduling: a case study, *Journal of Quality in Maintenance Engineering*, 5(2) , 78-88 (1999)
2. Alfares, H.K., A simulation approach for stochastic employee days-off scheduling, *International Journal of Modelling and Simulation*, 27(1), 9-15 (2007)
3. Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H., The state of the art of nurse rostering, *Journal of Scheduling*, 7(6), 441-499 (2004)
4. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D., Staff scheduling and rostering: A review of applications, methods, and models, *European Journal of Operational Research*, 153(1), 3-27 (2004)

5. Mason, A.J., Ryan, D.M., Panton, D.M., Integrated simulation, heuristic and optimisation approaches to staff scheduling, *Operations Research*, 46(2), 161-175 (1998)
6. Mundschenk, M., Drexel, A., Workforce planning in the printing industry, *International Journal of Production Research*, 45(20), 4849-4872 (2007)
7. Thompson, G.M., Labor staffing and scheduling models for controlling service levels, *Naval Research Logistics*, 44(8), 719-740 (1997)

An Investigation of a Genetic Algorithm and Sequential Local Search Approach for Curriculum-based Course Timetabling Problems

Salwani Abdullah • Hamza Turabieh • Barry McCollum • Edmund K Burke

1 Introduction

Curriculum-based course timetabling deals with the weekly assignment of a set of lectures for university courses to specific timeslots and rooms, where conflicts between courses are set according to curricula published by the university and not on the basis of enrolment data. The curriculum-based course timetabling is considered as the third track in the 2nd International timetabling competition (ITC2007). The main reason for the wide interest in this formulation is because it is capable of representing real world problems that often arise in higher educational institutions. In this paper, we consider the same curriculum-based course timetabling problem as described in Gaspero et al. [1]. The solution of the problem is an assignment of a period (day and timeslot) and a room to all lectures of each course subject to a set of hard and soft constraints. There are four hard constraints considered as in Gaspero et al. [1]:

- Lectures: All lectures of a course must be scheduled, and assigned to distinct periods.
- Conflicts: All lectures of courses in the same curriculum or taught by the same teacher must be scheduled in different periods.
- Availability: If the teacher of the course is not available to teach that course at a given period, then no lecture of the course can be scheduled at that period.
- Room Occupation: Two lectures cannot be assigned in the same room in the same period.

The soft constraints considered are also taken from Gaspero et al. [1]:

- Room Capacity: The number of students that attend the course for each lecture, must be less than or equal to the number of seats of all the rooms that host its lectures.

Salwani Abdullah
Center for Artificial Intelligence Technology, Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia.
E-mail: salwani@ftsm.ukm.my

Hamza Turabieh
Center for Artificial Intelligence Technology, Universiti Kebangsaan Malaysia, 43600 UKM Bangi, Selangor, Malaysia.
E-mail: hamza@ftsm.ukm.my

Barry McCollum
Department of Computer Science, Queen's University Belfast, Belfast BT7 1NN United Kingdom,
E-mail: b.mccollum@qub.ac.uk

Edmund K Burke
Automated Scheduling, Optimization and Planning Research Group, School of Computer Science & Information Technology, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, United Kingdom,
E-mail: ekb@cs.nott.ac.uk

- Min Working Days: The lectures of each course must be spread into the given minimum number of days.
- Isolated Lectures: Lectures belonging to a curriculum should be adjacent to each other (i.e., in consecutive periods).

2 The Algorithm

The algorithm approach discussed here is divided into two stages: initial construction and improvement. From the literature it can be observed that most of the initial construction solutions in relation to university timetabling problems are generated using graph coloring heuristics. This is directly related to how the problem in general is represented [7, 9]. Improvement is subsequently often achieved through the use of a metaheuristic approach. The following described the approach taken here.

2.1 Construction Algorithm

A construction algorithm proposed by Chiarandini *et al.* [2] and Landa-Silva and Obit [3] is used to generate large populations of random initial solutions. The constructive algorithm consists of three phases i.e. largest degree heuristic, neighbourhood search and tabu search as presented in pseudo code form in Fig. 1. We called this construction algorithm as a hybrid constructive algorithm. This approach was chosen in particular because it was able to produce feasible solutions for all datasets due to the combination of the strength from three phases involved (see [3]).

```

set population size, Popsiz
for i =0 to i < Popsiz
    for j=0 to j < Number of lectures
        Phase 1:Apply largest degree heuristic
    end for
    do while (timetable infeasible)
        Phase 2:Apply neighborhood search
        Phase 3:Apply tabu search
    end do
end for
return population of feasible timetables

```

Fig. 1 Pseudo code for construction algorithm

2.2 Improvement Algorithm

The improvement algorithm involves the implementation of genetic algorithm (GA) and sequential local search. GA was introduced by John Holland in 1975 [4], which is a family of computational models that employs processes found in the natural biological evolution. GA is an adaptive methods used to solve search and optimization problems. GAs encode a potential solution of a specific problem in a simple chromosome like data structure and apply recombination operators to these structures so as to preserve critical information. It is used to search large, nonlinear solution space where expert knowledge is lacking or difficult to encode. Moreover it requires no gradient information, evolves from one population to another and produces multiple optima rather than single local one. These characteristics make GA a well-suited tool for optimization problems as for example in [5] and [7].

Standard GA focuses on the global side of the searching area, while local search method mainly relies on local area. The more intensive the local search, the stronger the need of specialized information about the function to be minimized or maximized (depends on the problem to be solved). As a matter of fact, hybridization between GA and other local search methods has already been suggested and tested in an important number of works and they show the effectiveness of this combination (see [10], [11], [12], [13]). Due to this strength, our

improvement algorithm combines GA and sequential local search. Fig. 2 shows a schematic overview of the approach. The algorithm used in this paper in differ from [8] where there was no crossover operation employed and the algorithm was tested on 11 datasets as in Socha et al. [6]. At the beginning of the search, a set of populations is formed from the generated initial solutions. In each generation, multiple individuals are stochastically selected from the current population based on their fitness, recombined and mutated to form a new population, which becomes current in the next iteration of the algorithm. The repair function is applied to bring infeasible solutions to feasible once that works in three steps as below:

Step 1 : Find free timeslots for each room

Step 2 : Find free timeslots for each event

Step 3 : Find feasible timeslots for rooms and events (i.e. an intersection between Steps 1 and 2).

Sequential Local Search: Before moving to the next generation, we applied a neighbourhood search algorithm to improve the timetable. This process makes the convergence of the genetic algorithm faster. We then reinsert the solution of the local search back to the genetic algorithm to be considered in the next generation.

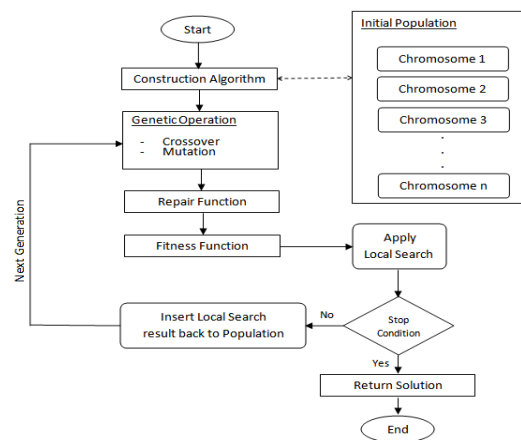


Fig. 2 A schematic overview of the approach

3 Experiments and Results

The experiments for the curriculum-based course timetabling problem discussed in this paper were tested on the twenty one real-world instances provided by the University of Udine. The details of all instances can be found in <http://tabu.diegm.uniud.it/ctt/index.php>.

Table 1 shows the parameters used in the genetic algorithm after preliminary experiments and, are comparable with the papers in the literature [7]. The proposed algorithm was programmed using Matlab and the simulations were performed on the Intel Pentium 4 2.33 GHz computer.

Table 1 Parameter Setting

Parameter	Generation Number	Population size	Crossover Rate	Mutation Rate	Selection Mechanism	Crossover Type
Value	10000	100	0.8	0.04	Tournament selection	Single point

Table 2 shows the results obtained and the comparison with best known solutions; the best results are presented in bold. The number of generations on each dataset is set to 10000 (with different random seeds). The best results out of 10 runs obtained are presented. Note that

our method produces the best known result in the literature on three of the twenty one problems¹. The processing time (considered as a stopping condition) for each run is set to 600 seconds based on the time allocate in the ITC2007 [1]. It can be seen that our approach is able to produce timetable with lowest penalty on comp01 and comp02 (tying with best known results) and also able to find global solution on comp11. At most of the cases, our results are comparable to the performance of tabu search and mathematical programming approaches. We believe that by introducing the intensification and diversifications strategies will help the algorithm to balance the converging process and then able to further improve the quality of the solutions.

Table 2 Results

Instance	Initial solution	GA and Sequential Local Search			Best Known Solution ¹	
		Best	Average	Time (s)	Result / Penalty	Method Used
comp01	1869	4	6.8	513.12	4	Tabu Search
comp02	6776	20	27.6	497.21	20	Tabu Search
comp03	6041	41	48.3	347.5	38	Tabu Search
comp04	4429	20	21.3	592.19	18	Tabu Search
comp05	7513	235	237.8	438.3	219	Tabu Search
comp06	4310	24	24	561.9	16	Mathematical Programming
comp07	3119	12	13.2	342.97	3	Mathematical Programming
comp08	3007	22	22.8	418.4	20	Mathematical Programming
comp09	4537	71	75.2	382.66	54	Tabu Search
comp10	2479	13	14.8	212.74	2	Mathematical Programming
comp11	1212	0	5.8	320.18	0	Tabu Search
comp12	3155	261	265.1	125.09	239	Tabu Search
comp13	4828	67	69.3	476.59	32	Tabu Search
comp14	3254	36	36	572.0	27	Tabu Search
comp15	5717	29	34.2	471.64	28	Tabu Search
comp16	4888	30	30	521.41	16	Tabu Search
comp17	3808	35	37.4	418.23	34	Tabu Search
comp18	1495	39	45.2	572.97	34	Tabu Search
comp19	4609	41	47.1	205.18	32	Tabu Search
comp20	5852	19	19	536.2	11	Tabu Search
comp21	4459	88	109.2	312.46	52	Tabu Search

Interested readers can find more details on instances and solutions that were contributed from the research community at <http://tabu.diegm.uniud.it/ctt/index.php>

4 Conclusion

This paper has focused on investigating a genetic algorithm and sequential local search for the curriculum-based course timetabling problem. Preliminary comparisons indicate that this algorithm is able to produce one global solution and two best known results. This is due to the hybridization of GA and sequential local search that able to embed an exploration and exploitation through the implementation of genetic operators within GA and sequential local search, respectively. Future research will be aimed at enhancing the performance of the local search and applying more genetic operators and tested on other real-world instances.

References

1. L Di Gaspero, A Schaerf and B McCollum, The Second International Timetabling Competition (ICT-2007) Curriculum-based Course Timetabling (Track 3), <http://pst.istc.cnr.it/ssc-at-icaps-07/papers/digaspero-et-al-SSC07.pdf> (2007).

¹ <http://tabu.diegm.uniud.it/ctt/index.php>

2. M Chiarandini, M Birattari, K Socha and O Rossi-Doria, An effective hybrid algorithm for university course timetabling. *Journal of Scheduling* 9(5), 403-432 (2006).
3. D Landa-Silva and JH Obit, Great Deluge with Nonlinear Decay Rate for Solving Course Timetabling Problems. Proceedings of the 2008 IEEE Conference on Intelligent Systems (IS 2008), IEEE Press, 8.11-8.18 (2008).
4. J Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press (1975).
5. EK Burke, D Elliman and RF Weare. A hybrid genetic algorithm for highly constrained timetabling problems. In *Proceeding of ICGA*, 605–610 (1995).
6. K Socha, J Knowles and M Samples, A max-min ant system for the university course timetabling problem. *Proceedings of the 3rd International Workshop on Ant Algorithms (ANTS 2002)*, Springer Lecture Notes in Computer Science Volume 2463, 1-13 (2002).
7. A Schaerf (1999), A survey of automated timetabling. *Artificial Intelligence Review* 13(2), 87-127 (2004).
8. S Abdullah and H Turabieh, Generating university course timetable using genetic algorithms and local search. The Third 2008 International Conference on Convergence and Hybrid Information Technology ICCIT, vol. I, 254-260 (2008).
9. E. Burke, J. Kingston and D. de Werra, Applications to Timetabling, *Handbook of Graph Theory*, (J. Gross and J. Yellen eds.), pp. 445-474, (Chapman Hall/CRC Press, 2003).
10. T. Goel and K. Deb. Hybrid methods for multi-objective evolutionary algorithms. In *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02)*, 188–192, (2002).
11. Uckun, S., Bagchi, S., Kawamura, K. and Y. Miyabe, 1993. Managing Genetic Search in Job Shop Scheduling". In *IEEE Expert Intelligent Systems and Their Application*, 15 – 24, (1993)
12. Wilke, P., Grobner, M., Oster, N.: A Hybrid Genetic Algorithm for School Timetabling. In: McKay, B., Slaney, J. (eds.) *AI 2002: Advances in Artificial Intelligence*, Springer-Verlag, New York, 455–464. (2002)
13. M. Rahoual and R. Saad, "Solving Timetabling Problems by Hybridizing Genetic Algorithms and Tabu Search", *Proc. 6th Int. Conf. on the Practice and Theory of Automated Timetabling*, Brno, Czech Republic, 467-472.,(2006)

Interactive Course Timetabling

Tomáš Müller · Hana Rudová · Keith Murray

1 Introduction

Timetabling problems are often solved based upon what is assumed to be complete information about the problem. However, users may later desire modifications to the generated timetable to adapt to changing requirements or increase their personal satisfaction with the solution [1,5,3]. Methods that allow corrections to an existing timetable while leaving most of the solution intact are necessary. Furthermore, the ability to make interactive changes to a timetable introduces a desirable approach by allowing users to readily react to a limited set of altered requirements by choosing among appropriate options. The aim of this paper is to introduce such methods of interactive timetabling as they have been applied at Purdue University. The concept of using interactive timetabling in this problem has been described previously [4], however, this work introduces a new formal view of this approach.

2 Applied Approach

Interactive changes to a timetable are easily understood by users as a sequence of changes to individual classes, Figure 1 illustrates the information available to a user considering changes to a class of interest (later denoted v_{bb}). The user may explore different options, consider various types of changes with the class, commit selected choices, or discard all changes considered. *Selected Assignments* describe changes already made to the timetable during the current interaction (denoted μ). *Conflicting Assignments* inform the user of any conflicts created in the timetable (denoted γ) as a result of the selected assignments. *Suggestions* (denoted Ω) are proposed options the user may choose from to make the timetable feasible.

T. Müller · K. Murray
Space Management and Academic Scheduling, Purdue University
400 Centennial Mall Drive, West Lafayette, IN 47907-2016, USA
E-mail: muller@unitime.org, kmurray@purdue.edu

H. Rudová
Faculty of Informatics, Masaryk University
Botanická 68a, Brno 602 00, Czech Republic
E-mail: hanka@fi.muni.cz

Current Assignment of MA 52700 Lec 2

Date:	Full Term
Time:	MWF 11:30a - 12:20p
Room:	EE 270
Initial Assignment:	MWF 11:30a - 12:20p EE 270
Student Conflicts:	3x BMS 82000 Lec 1 MWF 11:30a - 12:20p LYNN G167 [hard]
Room Locations:	23 (ARMS 1010, ARMS B061, BRNG 2280, CL50 224, ...)
Time Locations:	17 (MWF 7:30a, MWF 8:30a, MWF 9:30a, MWF 10:30a, ...)
Minimal Room Size:	118

Required Strongly Preferred Preferred Neutral Discouraged Strongly Discouraged Prohibited

Selected Assignments

Class	Date	Time	Room	Students
CE 37100 Lec 1	Full Term	MWF 8:30a	EE 270 → ARMS 1010	1→1 (h1→1)
MA 52700 Lec 2	Full Term	MWF 11:30a → MWF 8:30a	EE 270	3→9 (d0→9,h3→9)

Conflicting Assignments

Class	Date	Time	Room	Students	Constraint
AAE 33300 Lec 1	Full Term	MWF 8:30a → not-assigned	ARMS 1010 → not-assigned	-5 (h-3)	Room ARMS 1010

Not-assigned classes: +1 (0 → 1)

Overall solution value: +12.7 (7105 → 7117.7)

9x MA 52700 Lec 2 MWF 8:30a - 9:20a EE 270 [hard, distance]
 BME 69000 Lec 1 W 9:30a - 10:20a MJIS 1001
 1x CE 37100 Lec 1 MWF 8:30a - 9:20a ARMS 1010 [hard]
 POL 23700 Lec 1 MWF 8:30a - 9:20a ME 117

Assign

Suggestions

Score	Class	Date	Time	Room	Students
+10.4	AAE 33300 Lec 1	Full Term	MWF 8:30a → MWF 3:30p	ARMS 1010 → CL50 224	+138 (c+18,d+13,h+95)
	MA 16100 Lec 1	Full Term	MWF 3:30p → MWF 7:30a	CL50 224 → EE 129	
+24.6	AAE 33300 Lec 1	Full Term	MWF 8:30a → MWF 7:30a	ARMS 1010 → EE 129	+1 (d+9,h+3)
+25.3	AAE 33300 Lec 1	Full Term	MWF 8:30a → MWF 7:30a	ARMS 1010 → MATH 175	+1 (d+9,h+3)
+27.6	AAE 33300 Lec 1	Full Term	MWF 8:30a → MWF 7:30a	ARMS 1010 → WTHR 200	+11 (d+19,h+3)
	OLS 27400 Lec 1	Full Term	MW 7:30a	WTHR 200 → EE 129	

Fig. 1 Interactive solver interface for MA 52700 after selection of a new time assignment.

Figure 2 provides a more formal description of the interaction process leading to a desired solution. It is assumed that a solution δ of the timetabling problem P was generated by a standard timetabling solver such as that discussed by Murray et al. [4]. These become inputs for interactions with a user who may wish to explore changing the assignment of a class v_{bb} . The interactive solver BB generates a set of suggestions Ω along with conflicting assignments γ resulting from changes in v_{bb} and from any earlier selected assignments μ . Each suggestion $\omega \in \Omega$ represents a set of assignments resolving conflicts in γ .

The user has many possible responses, represented by the status variable S . When the user is satisfied with one of the suggestions ω , (s)he may *commit* the solution and the interactive process terminates. Any previously selected assignments μ are projected along with the new suggestions ω into δ . Alternatively, the user may suggest a specific assignment v_{bb}/d with the help of *selectAssignment(d)* to explore consequences of this

assignment by addition of v_{bb}/d to selected assignments μ . There is also a choice *selectClass*(c) to consider changes to a different class c , which becomes the new v_{bb} variable for future cycles. Note that the user can choose class c either from $\gamma \cup \mu$ or from Ω since these are available through the GUI. Another possibility, *removeClass*(c) allows deletion of a class c from μ and continuing with the reduced set of previously selected assignments. Finally, the user may decide to *abort* the interaction without applying any of the changes in μ to the original solution. If needed, (s)he can continue the interaction process by starting another interaction, e.g., selecting a different variable v_{bb} .

```

procedure INTERACTION( $P, \delta, v_{bb}$ )
   $\mu = \emptyset$ 
  while true do
    ( $\Omega, \gamma$ ) = BB( $P, \delta, \mu, v_{bb}$ )
     $S = \text{COMMUNICATION}(P, \delta, \mu, v_{bb}, \gamma, \Omega)$ 
    case ( $S$ ) commit( $\omega$ ):  $\delta = \text{JOIN}(\delta, \mu \cup \omega)$ ;  $\mu = \emptyset$ ; return
      selectAssignment( $d$ ):  $\mu = \mu \cup \{v_{bb}/d\}$ 
      selectClass( $c$ ):  $v_{bb} = c$  (only  $c \in \gamma \cup \mu$  or  $c$  from  $\Omega$  are available)
      removeClass( $c$ ):  $\mu = \mu \setminus \{c/d_c\}$ 
      abort: return
    end case
  end while
end procedure

```

Fig. 2 Cycle of interactive solving.

The timetabling problem P can be described as a constraint optimization problem [2] with domain variables, their domains, and constraints restricting the values of variables ($V, \mathcal{D}, C_h \cup C_s$). A solution δ is a complete assignment of variables satisfying all hard constraints C_h . The quality of a solution is evaluated based on violations of soft constraints C_s , e.g., with the help of weighted constraints expressing their importance. Interactive timetabling can also work with a partial consistent assignment ρ (satisfying constraints having all variables assigned) and extend it by selecting assignments for additional variables. When a user asks the solver BB for a solution to the problem for the variable v_{bb} and required assignments μ , the hard constraints C_h must be extended by the constraint $v_{bb} \neq d_o$ (if v_{bb}/d_o is present in ρ) and $v_{bb} = d$ (if required), and by the constraints $v_1 = d_1, \dots, v_k = d_k$ for $\mu = \{v_1/d_1, \dots, v_k/d_k\}$.

The constraint optimization problem becomes an input for the interactive solver (function BB at Figure 2). The solver is based on the branch and bound algorithm [2] with some extensions. Most importantly, the depth of the search is limited to allow changes to only a small number of variables (typically two or three), since the goal is only to create a new assignment for the variable v_{bb} . This limit may be extended and a subsequent call of BB invoked if needed. Second, a limited number n of best suggestions Ω is given to the user (note that the optimization criterion is computed through violations of weighted constraints and the best suggestions have the lowest value of weighted violations). As soon as the solution quality within current branch is worse than the quality of n -th found suggestion, its exploration is interrupted. Proper ordering of values for the assigned variables is necessary as is common for many other search algorithms. Each value is evaluated by its contribution to the value of the objective function and the best value is always selected. Finally, the search has a limit on the time (typically 5 seconds) in which to output a solution.

3 Experiments

To demonstrate the functionality of the branch and bound solver, it was applied on the largest problems at Purdue University, pu-fal07-llr and pu-spr07-llr. Characteristics of both problems, along with data sets and solutions, may be found at <http://www.unitime.org>, where the university timetabling application including an online demo is also available. Input solutions represent timetables used for the Fall and Spring semesters in 2007. Importantly, these timetables were generated by human schedulers using an automated solver [4] along with the interactive timetabling methods described here.

Suggestions were generated by the branch and bound solver for each class that was not fixed in time and room by a hard constraint. Changes to the assignments of up to two additional classes were allowed. Results in Table 1 are compared for runs with a time limit of 5 seconds (third and fifth columns) and runs without any time limit (second and fourth columns). The average time spent by the interactive solver roughly corresponds to the number of backtracks made. With the time limit, the solver was able to compute optimal solutions in more than half of the cases, with the quality of the solutions in the remaining cases being very good. Note that it was possible for the branch and bound solver to improve on the quality of input timetables since the solutions included manual changes by users having additional personal priorities.

Table 1 Results for the interactive solver.

Problem	pu-fal07-llr		pu-spr07-llr	
Classes	891		803	
Classes fixed in time & room (%)	31.0		33.8	
Time limit (s)	–	5	–	5
Optimal suggestion found (%)	98.4	51.5	99.2	67.0
Number of backtracks	66367.9	2886.9	13949.1	2592
Time spent (s)	128.6	4.7	39.9	4.2
Improvements in objective function (%)	+1.1	+0.8	+0.9	+0.7

4 Conclusions

A formal description of the approach applied to solving interactive timetabling problems at Purdue University has been presented. This technique was found to be a necessity by schedule managers who needed to make adjustments to automatically generated timetables as a result of changed requirements or to accommodate their own preferences. This allows creation of timetables that are ultimately used by the university.

Acknowledgements Hana Rudová was supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research intent No. 0021622419 and by the Grant Agency of the Czech Republic with grant No. 201/07/0205.

References

1. Hadrien Cambazard, Fabien Demazeau, Narendra Jussien, and Philippe David. Interactively solving school timetabling problems using extensions of constraint programming. In Edmund Burke and Michael Trick, editors, *Practice and Theory of Automated Timetabling V*, pages 190–207. Springer-Verlag LNCS 3616, 2005.
2. Rina Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.
3. Tomáš Müller and Roman Barták. Interactive timetabling: Concepts, techniques, and practical results. In Edmund Burke and Patrick De Causmaecker, editors, *PATAT 2002 — Proceedings of the 4th international conference on the Practice And Theory of Automated Timetabling*, pages 58–72, 2002.
4. Keith Murray, Tomáš Müller, and Hana Rudová. Modeling and solution of a complex university course timetabling problem. In Edmund Burke and Hana Rudová, editors, *Practice and Theory of Automated Timetabling VI*, pages 189–209. Springer-Verlag LNCS 3867, 2007.
5. Sylvain Piechowiak, Jingxua Ma, and René Mandiau. An open interactive timetabling tool. In Edmund K. Burke and Michael Trick, editors, *Practice and Theory of Automated Timetabling V*, pages 34–50. Springer-Verlag LNCS 3616, 2005.

Recent Work on Planning and Management of University Teaching Space

Andrew J. Parkes · Barry McCollum ·
Paul McMullan · Edmund K. Burke

1 Motivations and Overview of Existing Works

Universities have to invest considerable financial and human resources in the provision of space for teaching activities such as lectures, seminars, tutorials and workshops. Naturally, they would like such investments to be made wisely and efficiently. However, there is considerable evidence that in many Universities the resulting space is considerably underused. In a report by the Higher Education Funding Council for England (HEFCE), roughly speaking, it was found that often space was only used half the time, and then only half filled [9]. At least on the face of it this seems like an inefficient use of resources; it is natural to expect or hope that better planning for the space capacity would improve this. However, a fundamental problem has been that there was no real foundation for deciding whether or not such usage levels are in fact an inevitable result of meeting the timetabling and teaching resources, or alternatively, to provide methods to improve the situation. In this abstract we briefly overview our work towards remedying this situation. Here, we cannot hope to cover all work in the topic; instead the aim is to discuss our various strands of research and how they are woven together.

The space planning process needs to decide upon what space resources need to be provided for the envisaged teaching events. For this, an essential first step is to decide upon a measure of the quality of the provision. In reality, the true measure is likely to be a complex financial issue and highly specific to each institution. Hence, to make general progress, we follow standard practice and use the simpler measure of “utilisation”, which we take to simply be the average fraction (or percentage) of the seats used within the teaching space [5,6]. That is, it is a direct measure of the demand (by students) for a given supply (or space). So the (unfortunately common) situation that ‘half of the time rooms are empty, and are only half full when used’ corresponds to a utilisation of a paltry 25%.

Andrew J. Parkes (contact author), Edmund K. Burke,
School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK.
E-mail: { ajp, ekb }@cs.nott.ac.uk

Barry McCollum, Paul McMullan
School of Computer Science, Queen’s University, Belfast, BT7 1NN, UK.
E-mail: { b.mccollum, p.p.mcmullan }@qub.ac.uk

Firstly, it is important to observe that in a standard timetabling problem (see for example [11]) both the set of events and the rooms are fixed, hence, the utilisation is immediately and trivially derived from the problem description itself. The utilisation has the same value for all (feasible) solutions – standard timetabling software works to find feasible solutions and improve various quality measures, however, utilisation is not one of those measures. In particular, in order to be able to study the factors that influence utilisation we have to consider changes to at least one of the set of rooms (the supply) or the set of events (the demand): Our papers consider both of these changes.

Also, it is crucial to realise that space planning is inevitably performed without access to complete information about future teaching events and student enrolments. Space planning is an imperfect information problem; that is, a question of decision support (or optimisation) under uncertainty. This places it in stark contrast with the (vast majority) of timetabling research which works on classic perfect-information combinatorial optimisation problems. Hence, whilst space planning (in the sense we use) naturally includes timetabling issues, it is a separate problem in its own right; and substantially less developed than the timetabling arena. Note in the literature the term “space planning” might also be used, for example, finding the minimum space needed for a given set of events, and then this is a packing problem: Such planning is useful (and hard), however does not have the planning under uncertainty issues that inevitably arise in longer-term campus planning.

Achievement Curves:

The first step [5,6] was to define what we call “Achievement Curves” which are based on sampling different sets of events from a pool, and then attempting to assign as many as possible to rooms and timeslots. Each selected set of events corresponds to a “Requested Utilisation (U_R)” and the successfully assigned events gives a “Achieved Utilisation (U_A)”. Empirically, we observed that the statistical behaviour of U_A depends upon U_R in a fashion that exhibits a threshold (or phase transition) [8]. There was a “Critical Utilisation (U_C)” that separates two ‘phases’. In the under-constrained phase, $U_R < U_C$, with high probability all events can be scheduled. In the over-constrained phase, $U_R > U_C$, with high probability some events will inevitably remain unscheduled. Naturally, the actual value of the critical utilisation U_C is not universal, but depends on the particular room resources, characteristics of the set of events, details of the timetabling constraints, etc. However, we have essentially taken the value of U_C as a measure of the effectiveness of the resources compared to the demands, within the context of the specified constraints. Hence, improving utilisation is done on the basis of planning or modifying the set of resources so as to improve the critical utilisation. That is, the achievement curves themselves provide the basis of evaluation of a given set of room resources over a set of candidate events.

This initial work was done using only lecture events which are (generally) assigned as a single event. However, other teaching activities such as tutorials will often require splitting into smaller events. After such splitting into multiple sections, “student sectioning” then assigns students to sections so as to minimise timetable clashes (for recent work on this see [10]). The work on achievement curves was also applied to such cases [3, 1].

Computational Hardness Peaks:

Such thresholds are well known (at least within the field of artificial intelligence) to often be associated with strong peaks, and ‘easy-hard-easy’ patterns, in the computational difficulty of solving the typical problem instances [12]. In [4] we studied this behaviour and the resulting dramatic increase in computational effort needed as approaching the critical utilisation. We observed that this will have a practical impact; as potential utilisation improves then it is likely to need ever better timetabling software in order to fully exploit the potential. Poorer

algorithms might well be usable at low levels of utilisation but are likely to rapidly become impractical on improving space planning.

Room size profiles:

It is generally expected that to obtain good utilisation the distribution of sizes of the rooms needs, in some sense, to match the distribution of sizes of events. Conversely, a poor 'room size profile' will lead to poor utilisation. In practice, this can arise because of changes in teaching practices. A teaching facility might have been designed for the case of a few large lectures, but teaching practices changed to use many small lectures and tutorials. In [2], we provided methods to quantify this effect. This was then extended to provide some methods, based upon stochastic programming ideas, to design robust room size profiles. That is, to provide sets of target room capacities that would lead to improved critical utilisation values.

Space Type Mixing:

In teaching there are many different types of teaching activity (lectures, seminar, computer laboratories, etc), and each such event-type will have preferred space-type; that is, a type of room designed for a particular activity. In most work the event and space-types are treated individually and separately. However, in the real-world there are many interactions between space-types and often events will be placed in different space-type. In recent work [7] we have looked the effect of allowing such space-type mixing. A common problem is that the set of space-types becomes a poor match to the mix of event-types, and so also provided methods to select a better mix of space-types; that is, to provide a better 'space-type profile'.

2 Planned Work

Our existing work has looked at the many facets of space planning, but has generally treated them separately. Naturally, the most immediate goal is to combine all the work into an integrated approach. Also, of course, any such approach will need to be tested on a wide range of problems.

Finally, we note that there is a general and interesting issue about whether the timetabling model used within space planning needs to be a complete model or whether it suffices to use a simpler approximate model. We will investigate the question of whether some of the details needed when doing actual timetabling can be neglected when using timetabling within some system for space planning. That is, we will look at the extent to which some approximate timetable model can be used for space planning without causing a significant loss in the quality of the resulting space plans when presented with the complete timetabling model.

References

1. Camille Beyrouthy. *Models, Solution Methods and Threshold Behaviour for the Teaching Space Allocation Problem*. PhD thesis, School of Computer Science, University of Nottingham, 2008. (Submitted).
2. Camille Beyrouthy, Edmund K. Burke, Dario Landa-Silva, Barry McCollum, Paul McMullan, and Andrew J. Parkes. Improving the room-size profiles of university teaching space. Technical report, School of Computer Science, University of Nottingham, 2007. Presented at the Dagstuhl Workshop on 'Cutting, Packing and Space Allocation', March, 2007.
3. Camille Beyrouthy, Edmund K. Burke, Dario Landa-Silva, Barry McCollum, Paul McMullan, and Andrew J. Parkes. The teaching space allocation problem with splitting. In Edmund K. Burke and Hana Rudova, editors, *Revised Selected papers from the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, Brno, Czech Republic, volume 3867 of *Lecture Notes in Computer Science*, pages 228–247. Springer-Verlag, 2007.

4. Camille Beyrouthy, Edmund K. Burke, Dario Landa-Silva, Barry McCollum, Paul McMullan, and Andrew J. Parkes. Threshold effects in the teaching space allocation problem with splitting. *European Journal of Operational Research (EJOR)*, 2008. To appear.
5. Camille Beyrouthy, Edmund K. Burke, J. Dario Landa-Silva, Barry McCollum, Paul McMullan, and Andrew J. Parkes. Understanding the role of UFOs within space exploitation. In *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, pages 359–362, Brno, Czech Republic, 2006.
6. Camille Beyrouthy, Edmund K. Burke, J. Dario Landa-Silva, Barry McCollum, Paul McMullan, and Andrew J. Parkes. Towards improving the utilisation of university teaching space. *J Oper Res Soc*, 60(1):130–143, 2008.
7. Camille Beyrouthy, Edmund K. Burke, Barry McCollum, Paul McMullan, and Andrew J. Parkes. University space planning and space-type profiles. Submitted to *Journal of Scheduling*, 2007.
8. B. Bollobas. *Random Graphs*. Academic Press, London, England, 1985.
9. HEFCE. Estates management statistics project. Technical report, Higher Education Funding Council for England, March 1999. Report 99/18. http://www.hefce.ac.uk/pubs/hefce/1999/99_18.htm.
10. Keith Murray and Tomáš Müller. Real-time student sectioning. In Philippe Baptiste, Graham Kendall, Alix Murnier-Kordon, and Francis Sourd, editors, *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Application (MISTA 2007)*, Paris, France, pages 598–600, 2007.
11. Andrea Schaerf. A survey of automated timetabling. *Artif. Intell. Rev.*, 13(2):87–127, 1999.
12. Bart Selman, David G. Mitchell, and Hector J. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1-2):17–29, 1996.

An MILP-based approach to short-term production scheduling and lot sizing with major and minor setups

A case study from the beverage industry

Hans-Otto Günther

1 Introduction

As a case study, the production of beverages, e.g. fruit juices and soft drinks, is considered. In this industry, there is often only one single production stage after which final products are packed and shipped to warehouses or individual customers. This type of production environment is known as “make-and-pack production”. Specifically in the production of beverages, combined bottling and packaging lines are established for each package type, e.g. plastic bottles, carton boxes, and bag-in-box containers. A line usually produces a number of product types, e.g. juices of different flavor, and packages them into individual units for use by end customers. Each product type, e.g. orange or pineapple juice, corresponds to a specific recipe which determines the ingredients and the processing conditions of the product type.

To cope with the increased product variety, manufacturers mostly pursue a production policy with short production cycles for product types and frequent changeovers between products within the same product family. Typically, production lines are set up for a specific package form requiring a major setup such that the changeover between the individual product types can be accomplished with only a minor setup operation or cleaning out the equipment and the pipelines which connect the pre-mix tanks with the filling lines.

The intention of this paper is to develop an optimization model for short-term scheduling and lot sizing in beverage production. The model formulation is based on the block planning concept proposed in [1].

2. Block planning

Scheduling problems with sequence-dependent setup costs and times have been an area of considerable scientific interest (cf. the comprehensive reviews given in [2,3]). In many industrial applications, however, setup sequences are determined by technological restrictions. For instance, in industrial make-and-pack production systems, scheduling policies can be found which rely on a given sequence of product variants within a product family. For example, setups are sequenced from products with high to low purity requirements, from the lower taste of a food product to the stronger, or from the brighter color of a product to the darker. Such policies are known in practice as “block planning”. So far, in the scientific literature the application of block planning concepts and the development of corresponding optimization models are widely neglected though block planning concepts are easy to implement and reflect managerial practice prevalent in make-and-pack production. Only a few papers presenting ap-

Hans-Otto Günther

TU Berlin, Dept. of Production Management, Straße des 17 Juni 135, 10623 Berlin, Germany

E-mail: hans-otto.guenther@tu-berlin.de

plications of the block planning approach can be found, e.g. [1,4,5]. Major characteristics of the block planning concept are the following.

- Given the assignment of products to setup families, fixed setup sequences of products are defined based on human expertise and technological requirements.
- Each block corresponds to a setup family and is scheduled in a cyclical fashion, e.g. one block per week.
- The composition of the individual blocks is not necessarily the same. Binary decision variables indicate whether a product is set up or not and continuous decision variables reflect the lot size of each product in the block. Depending on the development of demand over time, the lot sizes of an individual product may vary from block to block. As a result, also the time needed to complete a block is variable.
- The start-off and completion dates of a block are not directly linked to the period boundaries. Hence, a block is allowed to start in an earlier period as soon as the predecessor block has been completed. However, the execution of a block must be finished before the end of the assigned period.
- Typically, a major setup operation is performed before starting or after completing a block (e.g. for cleaning the manufacturing equipment), while only a minor setup operation is required when changing between products within the same block (e.g. for provision of material or for adjusting the processing conditions).
- The usual objective function is to minimize total inventory holding and setup costs. Major constraints arise from the available production capacities and the satisfaction of external demand.
- Macro-periods, e.g. weeks, are used for the assignment of blocks while the assignment of external demand elements is based on micro-periods, e.g. days.
- For each product inventory balances are updated on a daily basis according to the production output from the various production lines and the given external demand.

3. Model formulation

Generally, the problem of determining production order sizes, their timing and sequencing in make-and-pack production with setup considerations is equivalent to a capacitated lot size problem. However, the related models presented in the academic literature do not sufficiently reflect the need for scheduling production orders on a continuous time scale with demand elements being assigned to distinct delivery dates. Moreover, issues like the definition of setup families with consideration of major and minor setup times and multiple non-identical production lines with dedicated product-line assignments are seldom addressed in a realistic way.

For application in make-and-pack production an MILP model based on the block planning concept is developed. It reflects major issues arising in the beverage industry, e.g. cyclical production policy, major and minor setups, clean-out of equipment at the end of the week, and daily demand assignments. The MILP model formulation uses a continuous time representation to model the production runs for all products within a setup family while a discrete time representation is used for the assignment of blocks to macro-periods and for the daily assignment of demand elements. Binary decision variables refer to the setup of blocks, production lots for packaging forms, and sub-lots for product types. Continuous variables are used to model the lot sizes and sub-lot sizes, inventory levels and the timing of the various production activities.

The objective function minimizes total production costs, clean-out costs at the end of a block, major setup costs for production lots, minor set up costs for sub-lots, and inventory holding costs. The MILP model contains a number of constraints in order to model the setup conditions, the allocation of production quantities between the production lot and the sub-lot level as well as the succession of production activities and setup operations on the bottling and packaging line. Since customer demands are assigned to daily delivery periods, a discrete heuristics function consisting of a sequence of binary decision variables is introduced in order to

trace the completion of production lots on a daily time scale. Through a switch from 0 to 1 these variables indicate that a specific production lot has been finished up to a particular day and the production output becomes available for delivery to customers or distribution centres.

4. Numerical experiments

The proposed optimization model was implemented on a standard PC using CPLEX as solver. Optimal solutions are obtained within a few seconds of CPU time for problem instances with a four-week planning horizon. Numerical results are carried out in order to compare a flexible block planning approach which allows a considerable time window for the execution of the production activities with rigid block planning which assumes narrow time limits for scheduling the individual blocks.

Data used in the numerical investigation reflect realistic industrial settings in the production of beverages. Since a short-term planning horizon of four weeks is considered, seasonal demand variations are reflected by two scenarios which assume an average workload of 75% and 90% of the available total capacity, respectively. Moreover, since demand in the fast moving consumer goods industry is driven by customer orders of various sizes, we consider the granularity of demand elements as a key factor in our numerical experiments. In order to reflect these issues, we examine three levels of demand granularity: high, medium, and low. For instance, high demand granularity implies a large number of small-sized customer orders while low demand granularity is given by a comparatively small number of large-sized orders. Numerical results clearly reveal the superiority of the flexible block planning approach.

References

- [1] H.-O. Günther, M. Grunow, U. Neuhaus, Realizing block planning concepts in make-and-pack production using MILP modelling and SAP APO[®], *International Journal of Production Research*, vol. 44, pp. 3711-3726, 2006.
- [2] X. Zhu, W.E. Wilhelm, Scheduling and lot sizing with sequence-dependent setup: A literature review, *IIE Transactions*, vol. 38, pp. 987-1007, 2006.
- [3] A. Allahverdi, C.T. Ng, T.C.E. Cheng, M. Kovalyov, A survey of scheduling problems with setup times and costs, *European Journal of Operational Research*, vol. 187, pp. 985-1032, 2008.
- [4] M. Lütke Entrup, H.-O. Günther, P. van Beek, M. Grunow, T. Seiler, Mixed integer linear programming approaches to shelf-life-integrated planning and scheduling in yoghurt production. *International Journal of Production Research*, vol. 43, pp. 5071-5100, 2005.
- [5] B. Bilgen, H.-O. Günther, Integrated production and distribution planning in the fast moving consumer goods industry: A block planning application. *OR Spectrum* (under second review), 2009.

Solving Berth Allocation Problem with Column Generation

Ceyda Oğuz · Özge Narin

Berth allocation problem is to find the best allocation of berths to the incoming ships in a container terminal. Container terminals are the regions that lay along the quayside where containers are transported from one place to another. Importance of the container terminals grows with the increasing need for transportation. The advantages of sea transportation can be realized better with efficiently operated container terminals. According to Imai et al. [1], influence of berth allocation has the highest impact on efficient operations of a container terminal. This is also supported by Vis and Koster [4], Steenken [2], and Murty [3] by showing that berths are the most important resources to operate a container terminal efficiently.

In Dynamic Berth Allocation Problem (DBAP) the ships will arrive during the planning horizon, which makes the problems much harder than the static berth allocation problem (SBAP). DBAP is solvable exactly up to 10 berths - 45 ships instance and there exists a good metaheuristic algorithm to handle larger instances [6]. We propose a column generation (CG) approach for solving DBAP and our motivation is to provide better solutions for large DBAP instances. Actually, this study is a preliminary study to assess the strengths and weaknesses of CG for DBAP. In the rest, we will briefly describe our CG approach and will present the preliminary results obtained for DBAP (Model used for DBAP is given in [5]).

CG is a good approach for those problems where the size grows immensely as a result of dealing with all the variables explicitly. CG generates only those variables which have the potential to improve the objective function. These variables, which have negative reduced costs, are determined by the pricing problem (subproblem). We have developed the CG based on the formulations given in [7] because of the similarity of that problem with DBAP.

Ceyda Oğuz
Koc University, Turkey
E-mail: coguz@ku.edu.tr

Özge Narin
Koc University, Turkey
E-mail: onarin@ku.edu.tr

We split the DBAP into two problems: the master problem and the subproblem (pricing problem). The master problem is the original problem with only a subset of variables being considered and the subproblem is a new problem created to identify a new variable. The objective function of the subproblem is the reduced cost of the new variable (column) with respect to the current dual variables, and the column should satisfy the constraints of DBAP.

For the master problem, all assumptions made in [5] for DBAP model remain the same and new assumptions are the following:

- (a) All berths have a feasible column set D_i indexed by $1, 2, \dots, d_i$.
- (b) Variable z_i^d denotes d^{th} column of berth i and if this variable is 1, cost of r_{id} is incurred for column $d=1, 2, \dots, D_i$ of berth $i=1, 2, \dots, I$.
- (c) Columns have T number of components for the ships denoted by $z_i^d = \{z_{i1}^d, z_{i2}^d, \dots, z_{iT}^d\}$. We define z_i^d as follows;

$$z_i^d = \begin{cases} 1 & \text{if column } d \text{ is chosen for berth } i \\ 0 & \text{otherwise} \end{cases}$$

If it is decided to schedule ship t at berth i in column d , z_{it}^d component of column d is 1, otherwise it is 0. So the master problem is;

$$\text{Minimize } \sum_{i \in B} \sum_{d \in D_i} z_i^d r_{id} \quad (1)$$

s. t.

$$\sum_{d \in D_i} z_i^d = 1 \quad \forall i \in B, \quad (2)$$

$$\sum_{i \in B} \sum_{d \in D_i} z_{it}^d z_i^d = 1 \quad \forall t \in V, \quad (3)$$

$$z_i^d \in \{0, 1\} \quad \forall i \in B, d \in D_i. \quad (4)$$

The linear relaxation of the master problem is solved at each iteration with the existing columns and after this solution, dual variables of the constraint set (3) are passed to the pricing problem as λ_j values. For the first iteration we need an initial solution to get dual variables to pass to the subproblem, which is found by applying a simple heuristic in a negligible time. During the iterations, once the stopping criteria is reached, the master problem is solved as a mixed integer programming model to obtain an integer solution.

In the subproblem we define;

$$a_{jk} = \begin{cases} 1 & \text{if ship } j \text{ is scheduled at position } k \\ 0 & \text{otherwise} \end{cases}$$

w_{jk} = the length of the idle period before the arrival of ship j that will be scheduled at position k .

Then the model is;

$$\text{Minimize } \sum_{j \in V} \sum_{k \in O} (kc_{ij} + S_i - A_j)a_{jk} + \sum_{j \in V} \sum_{k \in O} kw_{jk} - \sum_{j \in V} \sum_{k \in O} \lambda_j a_{jk} \quad (5)$$

s. t.

$$\sum_{k \in O} a_{jk} \leq 1 \quad \forall j \in V, \quad (6)$$

$$\sum_{l \in V} \sum_{m \in P_k} (c_{il}a_{lm} + w_{lm}) + w_{jk} - (A_j - S_i)a_{jk} \geq 0 \quad \forall j \in W_i, k \in O, \quad (7)$$

$$a_{jk} \in \{0, 1\} \quad \forall j \in V, k \in O, \quad (8)$$

$$w_{jk} \geq 0 \quad \forall j \in V, k \in O. \quad (9)$$

Subproblem is solved with the dual variables that come from the master problem. Objective function of the subproblem gives the reduced cost of the column generated for the berth at hand. Generated column is added to the master problem if reduced cost is negative. Since the generated column is for a specific berth, the subproblem is solved for each berth individually. Therefore the subproblem must be solved I (number of berth) times at each iteration.

We have tested CG approach on the data provided in [6] and compared its performance with the Variable Neighborhood Search (VNS) algorithm given in that study. Our solutions are obtained on an Intel(R) computer with Xeon(R) CPU 3.00 GHz, 4.00 GB of RAM. Ilog Concert Technology CPLEX 11.0 and Visual Studio 2005 are used for the solutions and the codes are written in C++. Our aim is to evaluate the CG approach in terms of the run time and solution quality.

The results are presented in Table 1, where error percentages and running times of VNS, CG, a heuristic CG (i.e. heuristic based subproblem) and, exact algorithms are given. Our results for DBAP show that for smaller instances, CG can find the optimal solution easily as VNS does. The superiority of CG becomes apparent for instances starting with 10 berths and 45 ships. For these instances CG is able to find the optimal solution whereas VNS cannot. We are able to solve instances of DBAP up to 10 berths and 50 ships.

CG and exact solution have the same solution quality but CPLEX is faster than CG for the instances that are solvable for it. When instances get larger and harder, CG becomes more important because the tree size of CPLEX solution cannot be coped due to the memory problem. Although we cannot find a solution with CPLEX for instances starting with 10 berths and 45 ships and larger, CG gives good solutions.

We are currently working on applying a heuristic method for the subproblem (pricing problem). Results received so far are given in the table at the end of this section in column *Heuristic CG*. This approach fairly decreased the run time of the CG algorithm without worsening the solution quality.

Table 1 DBAP instances

Berths	Ships	S_i^*	Error (Percentage)				Running time (Seconds)					
			VNS	CG	Heuristic	CG	Exact	VNS	CG	Heuristic	CG	Exact
10	45	1/2	0.08	0.00	0.00		(**)	3.54	16252.60	4661.11		(**)
		3/5	0.03	0.00	0.00		0.00	2.74	7634.95	1553.47		3250.55
		5/8	0.04	0.00	0.00		0.00	2.29	8203.13	1787.41		169.82
		7/8	0.00	0.00	0.00		0.00	0.36	2187.20	414.01		19.06
10	50	1/2	0.00	0.00	0.00		(**)	3.83	40669.23	17921.63		(**)
		3/5	0.03	0.00	0.00		(**)	2.80	21255.73	5284.85		(**)
		5/8	0.04	0.00	0.00		(**)	3.89	18301.43	6791.25		(**)
		7/8	0.01	0.00	0.00		0.00	1.63	3752.8	886.91		24.29

(*)Availability times of the berths, $S_i^* = 1/2, 3/5, 5/8, 7/8$, of the time interval between the arrivals of the first and last ships.

(**)Solution cannot be found because of “out of memory” error.

References

1. A. Imai, E. Nishimura, and S. Papadimitriou, “Berth allocation with service priority”, Transportation Research Part B, 37, 437-457 (2003)
2. D. Steenken, S. Voss, and R. Stahlbock, “ Container terminal operation and operations research - a classification and literature review”, OR Spectrum, 26, 3-49 (2004)
3. K.G. Murty, J. Liu, Y.-W. Wan, and R. Linn, “A decision support system for operations in a container terminal”, Decision Support Systems, 39, 309-332 (2005)
4. I.F.A. Vis, and R. de Koster, “Transshipment of containers at a container terminal:An overview”, European Journal of Operational Research, 147, 1-16 (2003)
5. P. Hansen and C. Oguz, “A note on formulations of the static and dynamic berth allocation problems”, Les Cahiers du GERAD G-2003-30, HEC Montréal (2003)
6. P. Hansen, C. Oguz, and N. Mladenović, “Variable neighborhood search for minimum cost berth allocation”, European Journal of Operational Research, 191, 636-649 (2008)
7. W.B. Powell, Z.-L. Chen, “Solving parallel machine scheduling problems by column generation”, INFORMS Journal on Computing, 11, 78-94 (1999)

Approximation algorithm for berth and quay crane allocation problem with bounds.

Jacek Błażewicz · Maciej Machowiak ·
Ceyda Oğuz

1 Introduction

We model the berth and quay cranes allocation problem (see, for example, [4,5]) as a moldable task scheduling problem. Moldable tasks form one type of parallel tasks that can be processed simultaneously on a number of parallel processors for which the processing times are a function of the number of processors assigned ([3]). If one makes the decision of how many processors to be used in executing the task at the time of making the scheduling decision, then we talk about moldable tasks. Hence the number of processors allocated to each moldable task is not part of the data but is a decision variable. However, once the schedule is given and implemented, the decision about the number of processors allocated to a moldable task cannot be changed (if one can change this decision after the scheduling decision has been made, in other words while the schedule is being executed, then such a task is called a malleable task).

We will take a different approach to incorporate the fact that the number of quay cranes allocated to a ship will affect its berthing time. Our approach is to consider the berth allocation problem together with the quay crane allocation and to model as a non-preemptable moldable task scheduling problem. In this model, we can represent the relationship between the processing time of a ship and the number of quay cranes allocated to it with different functions, which is not necessarily a linear function. This approach can simultaneously increase the utilization of quay cranes, shorten the turn-around time of ships, and decrease the waiting time of the containers.

Since, in real life certain ships may require at least a number of quay cranes and some ships may not be allocated with more than a certain number of quay cranes, in

Jacek Błażewicz
Institute of Computing Science, Poznan University of Technology, Poland
E-mail: jblazewicz@cs.put.poznan.pl

Maciej Machowiak
Institute of Computing Science, Poznan University of Technology, Poland
E-mail: mmachowiak@cs.put.poznan.pl

Ceyda Oğuz
Department of Industrial Engineering, Koç University, Turkey
E-mail: coguz@ku.edu.tr

our current model we will take a lower bound or an upper bound on the number of cranes to be allocated to a ship into account.

An introduction of a lower bound leads to better berth allocation of the ships in the case when length of a berth is limited. An upper bound depends on a length of a ship. We avoid situation in which to small ship many of the cranes have been allocated.

We would like to present an approximation algorithm which starts from the continuous version of the problem (i.e., where the tasks may require a fractional part of the resources) to solve discrete moldable problem with the bounds.

2 Problem formulation

The moldable task scheduling problem can be stated formally as follows: we consider a set of m identical *processors* (quay cranes) used for executing the set $\mathcal{T}$ of n independent, non-preemptable (i.e., once a task starts its execution, it has to be processed on the processors it is assigned to without any interruption for the period of its processing time) *moldable tasks* (ships). Each moldable task (MT) has a processing time which is dependent on the number of processors to be allocated for its execution. As the number of processors allocated to a task is a decision variable by definition of moldable tasks, we denote this processing time by $t_i(r)$, where r is the number of processors allocated to task T_i , $1 \leq r \leq m$. As a result, we define the *processing speed* of a moldable task which depends on the number of processors allocated to it. We let $f_i(r)$ denote the processing speed function for task T_i if r processors are used to execute this task. The dependence of a moldable task processing time on the number of processors allocated is given as a discrete function; that is, it takes values at the integer points only. In our current considerations we introduce two new values r^l and r^u being an lower and an upper bounds on the number of processors allocated, respectively. It means that only the processors allocation r_i to the task T_i , where $1 \leq r^l \leq r \leq r^u \leq m$, is allowed.

The criterion assumed is the schedule length, which is denoted by C_{max} with $C_{max} = \max_i \{C_i\}$, where C_i denotes the completion time of task T_i .

The problem of scheduling independent MT without preemption is NP-hard ([2]).

From a different point of view, we may consider a problem in which the processors represent a continuously divisible renewable resource bounded from above ([6]). In this problem the processor (resource) allocation is not required to be integer. In [1] it was shown that an optimal solution to such a problem in which processing speed functions are interpolated by piecewise linear functions between integer points (P-CNTN problem) can be found in polynomial time, assuming that the processing speed functions are all concave.

In the optimal solution for problem P-CNTN with the makespan value C_{cont}^* , there is a single processing interval $[0, C_{cont}^*]$ for each task and vector $\mathbf{r}^* = (r_1^*, \dots, r_n^*)$ represents the processor allocations for the tasks.

It is worth to emphasize, that solution of problem P-CNTN is a good starting point for the discrete models and obtained schedule length is absolute lower bound for the optimal discrete schedules. Thus, it can be use in our Moldable tasks scheduling problem.

3 Approximation algorithm for moldable tasks with constraints on processors allocation

In this section, we consider the problem of berth scheduling (as defined in Section 2). The discrete processing speed function is replaced by a continuous interpolation using piecewise linear functions between integer points. Then, we propose an algorithm to transform the schedule obtained from continuous version into a feasible schedule for the discrete moldable task model. In the proposed algorithm, a rounding scheme for a non-integer allocation is firstly described. Next, the tasks are thoroughly packed using several steps of rounding off in order to achieve a better utilization of the processors.

4 Computational experiments

As we mentioned, to construct a schedule for a non-preemptable moldable task scheduling problem, we first used the solution of continuous problem with concave and piecewise linear processing speed functions. Such approximation of these functions allows to obtain a processors allocation close to the optimal moldable solution. Furthermore, the allocation of the processors after rounding changes very rarely during the processing of the Algorithm.

We performed a set of computational experiments to test the average behavior of the algorithm for different types (shapes) of the processing speed functions. To evaluate the average behavior of the Algorithm, we used the following measure:

$$Relative\ error = C'_{max}/C^*_{cont}$$

where

- C'_{max} - the schedule length obtained by Algorithm,
- C^*_{cont} - the optimal schedule length of the continuous solution.

For m equal 30, we varied n to see the influence of changing the number of tasks on the average performance of the algorithm under different processing speed functions.

- (i) $f_i(r) = r^{\frac{1}{2}}$,
- (ii) $f_i(r) = 2 - 1/r$,
- (iii) $f_i(r) = \log_2(r + 1)$,

where $f_i(0) = 0, i = 1, \dots, n$ and $r = 1, \dots, m$

In the Figure 1 influence of different shapes of processing speed functions on *Relative error* is shown.

From the experiments conducted we see that the average relative error of the Algorithm does not exceed value 1.50 of the optimal schedule length for the discrete case.

References

1. Błażewicz, J., Kovalyov, M.Y., Machowiak, M., Trystram, D., Węglarz, J., Malleable task scheduling to minimize the makespan, *Annals of Operations Research*, 129, 65–80, (2004).
2. Du, J., Leung, J.Y-T., Complexity of scheduling parallel tasks systems. *SIAM Journal on Discrete Mathematics*, 2, 473–487, (1989).

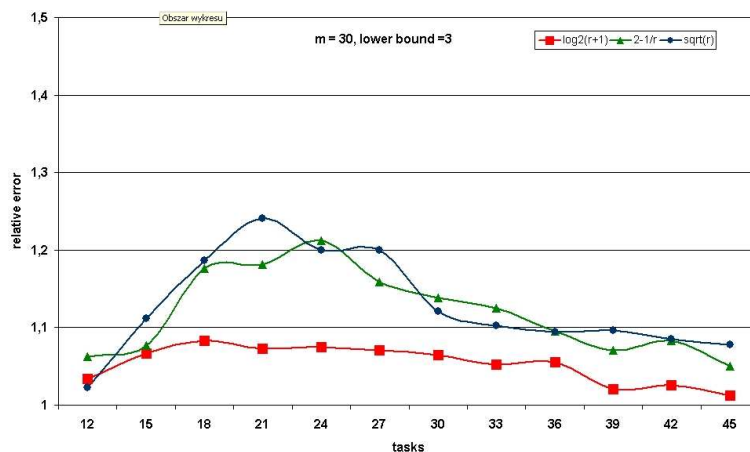


Fig. 1 Constant number of processors ($m=30$), lower bound = 3 for each task and varying number of tasks for different types of processing speed functions

3. Ludwig, W. T., *Algorithms for scheduling malleable and nonmalleable parallel tasks*, PhD thesis, University of Wisconsin - Madison, Department of Computer Sciences, (1995).
4. Park, Y-M., Kim, K.H., A scheduling method for berth and quay cranes, *OR Spectrum*, 25, 1-23, (2003).
5. Peterkofsky, R.I., Daganzo, C.F., A branch-and-bound solution method for the crane scheduling problem, *Transportation Research Part B*, 24, 159-172, (1990).
6. Węglarz, J., Modelling and control of dynamic resource allocation project scheduling systems, In S. G. Tzafestas (ed.), *Optimization and Control of Dynamic Operational Research Models*, North-Holland, Amsterdam, (1982).

Solving a dock assignment problem as a three-stage flexible flow-shop problem

Lotte Berghman · Roel Leus

1 Introduction

Toyota is one of the world's largest automobile manufacturers, selling over 7.5 million models (including Hino and Daihatsu) annually on all five continents, and generating almost 130 billion euro in net revenues. Since 1999, the total warehouse space floor of the European Distribution Centre TPCE (Toyota Parts Centre Europe) located in Diest (Belgium) has been expanded to 67.700 m². Toyota's Distribution Centre delivers to 28 European distributors on a daily basis.

At TPCE, the warehouse has some fifty gates, each with a capacity of one trailer, where goods can either be loaded on an empty trailer or be unloaded from a loaded trailer. Besides the warehouse with the gates, the site also has two parking lots, which can be seen as a buffer where trailers can be temporarily parked. All transportation activities of uncoupled trailers between these parking lots and the gates are done by two terminal tractors, which are tractors designed for use in ports, terminals and heavy industry. Obviously, if a trailer is coupled, there is no need for a terminal tractor since the trucker can transport the trailer. Because of the numerous loading, unloading and transportation activities, TPCE needs a schedule specifying the starting time and the assigned gate or terminal tractor for each activity.

Each arriving trailer, for which the planned arrival time is known, is either dropped off by the trucker at a parking lot and afterwards transported to a gate by the terminal tractor, or it is delivered immediately at a gate. After unloading or loading at the gate, the trailer is either picked up by a trucker, or transported to a parking space by a tractor, where it will be picked up by a trucker later on. In practice, every good has a fixed place in the warehouse and is preferably loaded or unloaded on a gate near this location.

The situation at TPCE can be modeled as a three-stage flexible flow shop. The first stage is the transportation from the parking lot to a specific gate, the second stage is the loading or unloading activity and the third stage is the movement back to a parking space. Two identical parallel machines execute both the first and the third stage. In these stages, the processing times are not dependent on the distance because the actual

Lotte Berghman, Roel Leus
ORSTAT, K.U.Leuven, Naamsestraat 69, 3000 Leuven, Belgium
E-mail: {lotte.berghman,roel.leus}@econ.kuleuven.be

driving time of the terminal tractor is small compared to the time it takes the driver to follow the safety instructions and attach the trailer to the tractor. In the first stage, the processing time is a non-zero constant when a task starts later than the ready time and in the third stage, the processing time is a non-zero constant when a task ends before the due date. Otherwise, the task is a dummy task with processing time equal to zero, because the tractor is not used. We will neglect the preference for certain gates in the remainder of this paper and model each gate as an identical machine; some fifty identical parallel machines execute the second stage. The processing time depends only on the job and is independent of the machine.

2 Parallel machines

When we only look at the second stage and do not take the terminal tractors into account, the problem at hand is close to $P_m|r_j, d_j|\sum w_j T_j$ in the standard three-field notation, although some tasks have deadlines rather than due dates. We define T_2 with $|T_2| = n$ as the set of all unloading and loading activities. T_2^U, T_2^L is a partition of T_2 : T_2^U is the set of unloading tasks, while T_2^L is the set of loading tasks. All tasks $t \in T_2$ have a ready time r_t . For the unloading tasks $t \in T_2^U$, this ready time equals the planned arrival time of the trailer. For the loading tasks $t \in T_2^L$, $r_t = 0$ because we assume that all the goods to be loaded on the trailers are in the warehouse and that the empty trailer is on site. All tasks $t \in T_2$ have a weight w_t . All unloading tasks $t \in T_2^U$ have a due date d_t equal to their ready time because the aim is to avoid tardiness. All loading tasks $t \in T_2^L$ have a deadline $\bar{d}_t$ based on the estimated driving time to the customer and the agreed arrival time at the customer. The number of identical parallel machines in stage two is $m < n$. Each machine can process at most one job at a time. The processing time p_t of a job $t \in T_2$ is the time needed to load or unload the trailer at the gate. Our problem consists of scheduling the tasks in such a way that the total weighted lateness (or tardiness, since $d_t = r_t$) of the unload operations minus the total weighted earliness of the load operations is minimized.

We will use mathematical programming to solve this problem. The first formulation tested is based on Dessouky [3], although Dessouky's objective is to minimize the maximum lateness. Tasks are assigned to positions at gates and additional sequencing variables are introduced to linearize the model. The job starting times constitute an extra set of decision variables.

Our second formulation has much in common with the ones presented by Jain and Grossman [6] and by Sadykov and Wolsey [8], where the objective is to minimize the summed task-to-machine assignment costs. The main difference with the first formulation is that individual machine positions are no longer decided upon, but we directly establish a (transitive) predecessor relation between jobs on the same machine.

The third model is based on Bard and Rojanasoonthon [1]; our setting is dissimilar from [1] in that the reference considers two time windows per task, setup times and priority classes. We let each task be the immediate successor of another task on the same machine and add dummy tasks as the first task on each machine.

The final formulation is a time-indexed model based on Dyer and Wolsey [4], van den Akker et al. [10] and Bigras et al. [2]; contrary to the foregoing three continuous-time formulations, we now no longer need starting times as separate decision variables.

3 Three-stage flexible flow-shop

In the second part of this work, we introduce the terminal tractors as the resources that are shared by stages one and three. Each job is composed of three tasks, one for each stage, and each task has to be scheduled on exactly one machine in the appropriate stage. The facilities are disjunctive, in the sense that each machine may process only one task at a time. Task preemption is not allowed.

We let T represent the set of all tasks; $|T| = 3n$. Set T is partitioned into T_1, T_2, T_3 . The set T_1 contains all transportation activities from the parking lot to a gate, T_2 comprises all tasks on the gates (as in Section 2) and T_3 are the movements from a gate to the parking lot. Each set T_i ($i = 1, 2, 3$) is further partitioned as T_i^U, T_i^L , with these two sets holding the unload and load tasks, respectively.

All tasks $t \in T_1$ have a ready time r_t . For the unloading tasks $t \in T_1^U$, this ready time equals the planned arrival time of the trailer. For the loading tasks $t \in T_1^L$, $r_t = 0$ because we assume that all the goods to be loaded are present in the warehouse and that the empty trailer is available. All tasks $t \in T_2^U \cup T_3^L$ have a weight w_t . All unloading tasks $t \in T_2^U$ have a due date d_t equal to the ready time of the corresponding task $t \in T_1^U$ because the aim is to avoid tardiness. All loading tasks $t \in T_3^L$ have a deadline $\bar{d}_t$ based on the promised arrival time at the customer. All transportation activities between the parking lot and the gates have a constant duration C , independent of the distance. The remainder of the problem description, including the objective function, is similar to the one presented in Section 2.

The mathematical formulations that are listed in Section 2 will be extended to incorporate the planning of stages one and three.

4 Implementation

In our presentation, we will report extensive computational results on benchmark datasets, both for the parallel-machine case as well as for the three-stage flexible flow-shop problem. We also provide some insights into the differences in computational efficiency between the formulations tested.

A straightforward implementation of the mathematical-programming formulations allows the exact solution within reasonable CPU time of medium-sized scheduling instances only. In order to increase the number of tasks that can be scheduled, we investigate the use of Lagrangian relaxation. Lagrangian relaxation is an approach to handle computationally hard optimization problems, in which some sets of difficult constraints are dualized to create a relaxed problem that is easier to solve. Upper bounds, corresponding with feasible solutions, can usually be obtained via minor modifications of the Lagrangian solution [5,7].

Competitive solutions for parallel-machine scheduling based on mathematical programming often rely on Dantzig-Wolfe decomposition in combination with a branch-and-price algorithm, see for instance Bard and Rojanasoonthon [1] and Sadykov and Wolsey [8]. The problem is reformulated as a master problem and m subproblems, one per machine, where the master problem selects schedules and the subproblems search for promising (partial) schedules. van den Akker et al. [9] also find the best set of mutually distinct machine schedules that contain all jobs, where new columns are generated by a local-search procedure. These approaches are very difficult to extend

to a flexible flow-shop setting, however, because the single-machine schedules are now interdependent due to the joint use of the terminal tractors.

References

1. Bard J.F. and Rojanasoonthon S., A branch-and-price algorithm for parallel machine scheduling with time windows and job priorities, *Naval Research Logistics*, 53, 24 – 44 (2006)
2. Bigras L., Gamache M. and Savard G., Time-indexed formulations and the total weighted tardiness problem, *INFORMS Journal on Computing*, 20, 1, 133 – 142 (2008)
3. Dessouky M.M., Scheduling identical jobs with unequal ready times on uniform parallel machines to minimize the maximum lateness, *Computers and Industrial Engineering*, 34, 4, 793 – 806 (1998)
4. Dyer M.E. and Wolsey L.A., Formulating the single machine sequencing problem with release dates as a mixed integer problem, *Discrete Applied Mathematics*, 26, 255 – 270 (1990)
5. Fisher M.L., An applications oriented guide to Lagrangian relaxation, *Interfaces*, 15, 2, 10 – 21 (1985)
6. Jain V. and Grossman I.E., Algorithms for hybrid MILP/CLP models for a class of optimization problems, *INFORMS Journal of Computing*, 13, 258 – 276 (2001)
7. Möhring R.H., Schulz A.S., Stork F. and Uetz M., Solving project scheduling problems by minimum cut computations, *Management Science*, 49, 3, 330 – 350 (2003)
8. Sadykov R. and Wolsey L.A., Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates, *INFORMS Journal on Computing*, 18, 209 – 217 (2006)
9. van den Akker M., Hoogeveen H. and van Kempen J., Parallel machine scheduling through column generation: minimax objective functions, release dates, deadlines, and/or generalized precedence constraints, Technical Report UU-CS-2005-027, Institute of Information and Computing Sciences, Utrecht University, The Netherlands (2005)
10. van den Akker J.M., Hurkens C.A.J. and Savelsbergh M.W.P., Time-indexed formulations for machine scheduling problems: Column generation, *INFORMS Journal on Computing*, 12, 2, 111 – 124 (2000)

Project scheduling with alternative technologies and stochastic activity durations

Stefan Creemers · Bert De Reyck · Marc Lambrecht · Roel Leus

1 Introduction

Research and Development (R&D) companies, whose core business relies on innovation, face a daily challenge of planning and surviving in an uncertain environment. Not only do uncertainties exist regarding the benefits and costs that their business entails, but they must also bear the risk of technical failure of their R&D activities.

In this paper, we focus on a single firm facing an R&D project with many possible development patterns. The project is composed of a collection of R&D activities with well-defined precedence relationships, where each activity is characterized by a cost, a duration and a probability of success. The successful completion of an activity corresponds to a technological discovery or a scientific breakthrough. Additionally, for obtaining certain results, more than one alternative may exist, and these alternatives can be pursued either in parallel or sequentially; the decision maker might also have discretion over the selection of the alternatives to pursue.

The objective is to schedule the activities in such a way as to maximize the expected net present value (NPV) of the project, taking into account the activity costs, the cash flows generated by a successful project, the stochastic activity durations and the probability of failure of each of the activities.

2 Definitions and problem statement

A project consists of a set of modules $M = \{0, 1, \dots, m\}$; each module $i \in M$ is a set of activities N_i that pursue a similar target. Activities should be executed without interruption. We denote the set of all activities by $N = \bigcup_{i \in M} N_i$ and let $n = |N| - 1$. A is a (strict) partial order on M , i.e. an irreflexive and transitive relation, which represents technological precedence constraints. (Dummy) modules 0 and m represent

Stefan Creemers, Marc Lambrecht, Roel Leus
K.U.Leuven
E-mail: <firstname>.<lastname>@econ.kuleuven.be

Bert De Reyck
University College London and London Business School
E-mail: bdeyck@london.edu

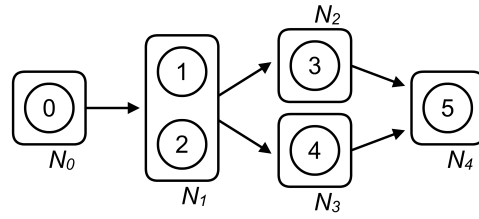


Fig. 1 Example module network

the start and the end of the project, respectively; they are the (unique) least and greatest element of the partially ordered set (M, A) , and are assumed to contain only one (dummy) activity, indexed by 0 and n , respectively. On the activities within each module i , we also impose a partial order B_i . For an illustration of these definitions, we refer the reader to Figure 1. The project consists of six activities, $N = \{0, 1, 2, 3, 4, 5\}$, where 0 and $n = 5$ are dummies. There are five modules, so $m = 4$: $N_0 = \{0\}$, $N_1 = \{1, 2\}$, $N_2 = \{3\}$, $N_3 = \{4\}$ and $N_4 = \{5\}$. In this case, B_1 is empty.

The duration D_j of each activity j is a random variable (r.v.); we denote the vector $(D_0, D_1, \dots, D_n)$ by $\mathbf{D}$. In the remainder of this paper, we assume the durations of the activities $j \in N \setminus \{0, n\}$ to be mutually independent exponentially distributed r.v.s with mean $E[D_j] = \mu_j^{-1}$ and $\mu_j > 0$; for the dummies, $D_0 = D_n = 0$. In the literature, an activity network with independent and exponentially distributed durations is often referred to as a Markovian PERT network.

We focus on project scheduling with NPV objective. A cash flow c_j is associated with each activity $j \in N \setminus \{n\}$, which is a real number that is assumed to be negative in the context of this paper. Each cash outflow is incurred at the start of the activity. Each activity $j \in N \setminus \{n\}$ also has a probability of technical success (PTS) p_j ; we assume that $p_0 = 1$. We consider the outcomes of the different tasks to be independent and assume that information on activity success and duration becomes available only at the end of the activity. Overall project success generates an end-of-project payoff $C \geq 0$, which (if the project succeeds) is received at the start of activity n . This final project payoff is only achieved when all modules are successful. Module $i \in M$ is said to be successful if at least one of its constituent activities succeeds. An activity $i \in N_j$ can be started only when success is obtained for all the predecessor modules of the module j to which i belongs. In order to account for the time value of money, we define r to be the applicable continuous discount rate: the present value of a cash flow c incurred at time t equals ce^{-rt} . Without loss of generality, we assume that C is large enough for the project to be initiated.

A solution to the scheduling problem described is a *policy*, which defines *actions* at *decision times*. Decision times are typically $t = 0$ (the start of the project) and the completion times of activities; a tentative next decision time can also be specified by the decision maker. An action entails the start of a new set of activities. Our goal in this text is to select a policy that minimizes the expected NPV. The generality of this problem statement suggests that optimization over the class of all policies will probably turn out to be computationally intractable. We therefore restrict our optimization to a subclass that has a simple combinatorial representation and where decision times are limited in number: our solution space consists of all policies that start activities only at the end of other activities (activity 0 is started at time 0).

3 Markov decision chain

For Markovian PERT networks, Kulkarni and Adlakha [4] describe an exact method for deriving the distribution and moments of the earliest project completion time using continuous-time Markov chains (CTMCs). Buss and Rosenblatt [1], Sobel et al. [5] and Creemers et al. [2] investigate the NPV objective and use the CTMC described in [4] as a starting point for their algorithm. In this work, we incorporate the concepts of activity failures and modular completion of a project into this procedure.

At any time instant t , an activity's status is either *idle* (= unstated), *active* (= in the process of being executed), or *past* (= successfully finished, failed, or considered redundant because its module is completed); we write $\Omega_j(t) = 0, 1$ or 2 , respectively, for $j \in N$. The state of the system is defined by the status of the individual activities and is represented by vector $\Omega(t) = (\Omega_0(t), \Omega_1(t), \dots, \Omega_n(t))$. State transitions take place each time an activity becomes past and are determined by the policy at hand. The project's starting conditions are $\Omega_0(0) = 1$ and $\Omega_j(0) = 0, \forall j \in N \setminus \{0\}$, the completion conditions state that $\forall j \in N : \Omega_j(t) = 2, \forall t \geq \omega$, where ω indicates the project completion time. Three different state transitions can occur: (1) activity $j \in N_i$ completes successfully; (2) activity $j \in N_i$ fails and some activity $k \in N_i$ is still idle or active; (3) activity $j \in N_i$ fails and all other activities $k \in N_i$ have already failed.

The problem of finding an optimal scheduling policy for the problem defined in Section 2 corresponds to optimizing a discounted criterion in a continuous-time Markov decision chain (CTMDC) on the state space Q , with Q containing all the states of the system that can be visited by the transitions. We apply a backward stochastic dynamic-programming (SDP) recursion to determine optimal decisions, in line with [2,5] and based on the CTMC as described in [4], and we ensure efficient memory management by means of an extension of the state-space partitioning proposed in [2].

4 Example

We consider an example project with $n = 5$, for which the module order A is described by Figure 1. Further input data are provided in Table 1; the project's payoff value C is 450 and the discount rate $r = 10\%$ per time unit.

At the onset of the project (at state $(2, 0, 0, 0, 0, 0)$) we can decide to start either the first activity, the second activity, or both. The SDP recursion evaluates the expected outcome of each of these decisions and selects one that yields the highest expected NPV (assuming optimal decisions are made at all future decision times). In our example, it is optimal to start only the first activity (corresponding with an objective function of 14.70) and we subsequently end up in state $(2, 1, 0, 0, 0, 0)$, in which two possibilities arise. If activity 1 succeeds, module 1 succeeds as well and a transition occurs to state

Table 1 Project data for the example project

task i	cash flow c_i	mean duration $E[D_i]$	PTS p_i
0	0	0	100%
1	-90	10	75%
2	-60	2	25%
3	-5	2	100%
4	-5	2	85%

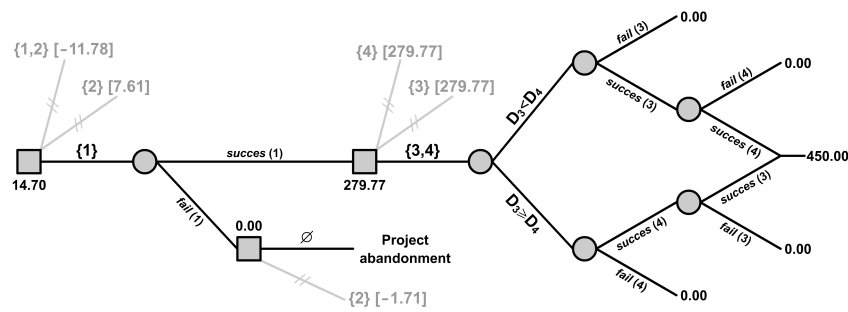


Fig. 2 Optimal paths in the decision tree for the example project

(2, 2, 2, 0, 0, 0); otherwise (if activity 1 fails), we end up in state (2, 2, 0, 0, 0, 0) and have to make a decision: either we start activity 2, corresponding with a transition to state (2, 2, 1, 0, 0, 0) and a net present value at that time for the remainder of the project of -1.71 , or we abandon the project altogether obtaining a current value of 0. The optimal decision in this case is obviously not to continue the project.

After a successful completion of module 1, two new activities become eligible and $2^2 - 1$ (i.e., not counting the decision not to start any activities) decisions are to be evaluated. The optimal decision is to start both the activities 3 and 4, leading to state $(2, 2, 2, 1, 1, 0)$. With two activities in progress, two possibilities arise: either activity 3 or activity 4 finishes first. Additionally, in each of these two cases, we need to account for the event of activity failure. Irrespective of which activity completes first, if either activity 3 or 4 fails then the entire project fails and the payoff is not obtained. If activity 3 (resp. 4) finishes first and succeeds, activity 4 (resp. 3) is still in progress and needs to finish (successfully) for the project payoff to be earned.

The relevant part of the corresponding decision tree is illustrated in Figure 2, in which the project evolves from left to right. A *decision node*, represented by a square $\square$, indicates that a decision needs to be made at that point in the process; a *chance node*, denoted by a circle $\bigcirc$, indicates that a random event takes place. Below each of the decision nodes, we indicate the expected NPV at the decision moment, which applies only to the part of the project that remains to be performed. For each decision node, a double dash // is added to each branch that does not correspond with a best choice in the associated SDP recursion.

The optimal policy was described in a forward manner in the previous paragraphs. Our SDP recursion proceeds backward, however, starting in state $(2, 2, \dots, 2, 1)$. Subsequently, the value function is computed stepwise for all other states. As the algorithm progresses, the information on the earlier generated states will no longer be required for further computation and therefore the memory occupied can be freed; this procedure is based on a partition of Q , where the necessary subsets are generated and deleted when appropriate.

5 Performance

We have used *RanGen* [3] to create project network instances of varying size (n) and order strength (OS), the latter being a measure for the density of the network: OS is the

proportion of precedence-related activity pairs compared to the theoretically maximum number of such pairs. These project networks were altered by gathering activities into modules. Preliminary computational results indicate that for $OS = 0.4$, we are able to analyze networks of up to 50 activities. With respect to $OS = 0.6$ and $OS = 0.8$, instances with up to respectively 80 and 120 activities are successfully solved.

References

1. Buss, A.H. and M.J. Rosenblatt (1997). Activity delay in stochastic project networks. *Operations Research* **45**, 126–139.
2. Creemers, S., R. Leus and M. Lambrecht (2008). Scheduling Markovian PERT networks with maximum-NPV objective. *Working Paper 0811, Department of Decision Sciences and Information Management, FBE, K.U.Leuven, Leuven, Belgium*.
3. Demeulemeester, E., M. Vanhoucke and W. Herroelen (2003). A random network generator for activity-on-the-node networks. *Journal of Scheduling* **6**, 13–34.
4. Kulkarni, V.G. and V.G. Adlakha (1986). Markov and Markov-regenerative PERT networks. *Operations Research* **34**, 769–781.
5. Sobel, M.J., J.G. Szmerekovsky and V. Tilson (2009). Scheduling projects with stochastic activity duration to maximize expected net present value. *European Journal of Operational Research* **198**, 697–705.

Time-constrained project scheduling with adjacent resources

J.L. Hurink · A.L. Kok · J.J. Paulus ·
J.M.J. Schutten

1 Extended Abstract

We develop a decomposition method for project scheduling problems with adjacent resources. Adjacent resources are resources for which the units assigned to a job are required to be in some sense adjacent. Possible examples of adjacent resources are dry docks, shop floor spaces, and assembly areas. We focus on the Time-Constrained Project Scheduling Problem (TCPSP), [4], with one 1-dimensional adjacent resource. However, the presented concepts and methods can be easily extended to more general models, e.g. multiple 1-dimensional adjacent resources or 2-dimensional adjacent resources.

The Time-Constrained Project Scheduling Problem (TCPSP) with an adjacent resource is defined as follows. We are given a set of jobs, a set of renewable resources and one 1-dimensional adjacent resource. Each job is characterized by a release date, processing time, deadline and its resource requirements, and has to be scheduled without preemption. The processing of jobs is further restricted by precedence relations. The adjacent resource is a special type of resource that is characterized by two properties. First, the resource units of adjacent resource are somehow topologically ordered (in this case ordered on a line) and the resource units assigned to a job have to be neighbored/adjacent and reassignment is not allowed. Second, motivated by the occurrence of adjacent resources in real life problems, we consider the more general case that the adjacent resource is not required only by a single job but by groups of jobs (called job

J.L. Hurink
University of Twente
E-mail: j.l.hurink@utwente.nl

A.L. Kok
University of Twente
E-mail: a.l.kok@utwente.nl

J.J. Paulus
University of Twente
E-mail: j.j.paulus@utwente.nl

J.M.J. Schutten
University of Twente
E-mail: j.m.j.schutten@utwente.nl

groups or simply groups). As soon as a job of such a job group starts, the assigned adjacent resource units are occupied, and they are not released before all jobs of that group are completed. In the considered model, it is only possible to hire additional capacity for the renewable resources, and not for the adjacent resource. The objective is to find a feasible assignment of the job groups to the adjacent resources and a feasible job schedule that minimizes the cost of hiring additional capacity.

The consideration of adjacent resources in the above mentioned form is motivated by a cooperation with a Dutch consultancy company. They encountered at several of their clients adjacent resource requirements. Since the project scheduling models in the literature do not cover these requirements, the company either assigns the adjacent resources in advance based on simple rules or they relax the adjacency requirements and repair the achieved solutions afterwards. However, since both approaches do not lead to satisfactory solutions, the company strives to incorporate adjacent resources in their planning software for project scheduling. One practical application is from the ship building industry that we use to illustrate the adjacency requirements. In this problem the docks form 1-dimensional adjacent resources, and all jobs related to building a single ship form a job group. Clearly, the part of the dock assigned to one ship has to satisfy the adjacency requirement. As soon as the construction of a ship starts, the assigned part of the dock is occupied until the construction is finished and the ship is removed from the dock. Removal or repositioning of a partially assembled ship is in practice too cumbersome and time consuming and therefore not an option. The other resources required to build the ships (like machines, equipment and personnel) can be modeled as renewable resources. The capacity of the dock is fixed but the capacity of renewable resources can be increased, e.g. by hiring additional personnel.

Adjacent resources have some relation with other special resource types considered in the literature. Spatial resources, as introduced in [2], are also resources which are not only required by a single job but by a group of jobs. However, no adjacency of the assigned resource units is required. Make-to-order assembly problems under assembly area constraints, see e.g. [6,7], form a special case of project scheduling problems with spatial resources where each job group requires exactly one unit of the spatial resource. In this case the adjacency requirement is automatically fulfilled. Without the adjacency requirement on the resources, the spatial resource can also be modeled with the concept of cumulative resources, see e.g. [1,9,10]. Cumulative resources are, for example, used to incorporate storage facilities into project scheduling problems. When a job group starts the cumulative resource is depleted by a given amount, and replenished as soon as a job group completes. In this paper we show why an adjacent resource cannot be modeled as a cumulative resource.

The literature that does consider an adjacency requirement on resources, only considers the special case in which exclusively adjacent resources are considered and groups consist of a single job. In this case the scheduling problem can be seen as a 2-dimensional packing problem. Examples of this can be found in literature on berth allocation at container terminals, reconfigurable embedded platforms, and check-in desks at airports. In [5] such packing problems are modeled by introducing a mode for each possible placement of a job on the adjacent resource. Consequently, one has to solve a multi-mode project scheduling problem with possibly an exponential number of modes.

Relaxing the group and adjacency requirements, the considered problem reduces to the TCPSP as considered in [4]. The study of such types of time-constrained project scheduling problems started with [8] and [3], for an overview see [10].

Summarizing, the concepts of job groups and adjacency requirement on resources have been treated in the literature, but never in a combined manner. To the best of our knowledge this work is the first to consider this combination.

In this paper we first formally state the time-constrained project scheduling problem with one 1-dimensional adjacent resource and discuss in why existing modeling and solution techniques for related problems are not applicable when we are dealing with an adjacent resource. Next we describe a decomposition method, the main contribution of this paper. This approach, first assigns the groups to the adjacent resource and then schedules the jobs. The group assignment problem is solved via an ILP model. Each solution of the group assignment problem implies additional precedence relations between the jobs. Once these precedence relations are added, the scheduling of the jobs can be done with a method for the TCPSP, e.g. the method of [4]. To improve the quality, we introduce objective functions for the group assignment problem in order to steer the assignment to a promising one. Computational tests are presented to indicate the potentials of the presented approach.

Acknowledgements Part of this research has been funded by the Dutch BSIK/BRICKS project.

References

1. J.C. Beck, Heuristics for scheduling with inventory: Dynamic focus via constraint criticality, *Journal of Scheduling*, 5(1), 4369 (2002)
2. De Boer, R., Resource-Constrained Multi-Project Management, A hierarchical decision support system, PhD Thesis, University of Twente, (1998)
3. Deckro, R.F. and Herberti, J.E., Resource constrained project crashing, *OMEGA International Journal of Management Science*, 17(1), 6979 (1989).
4. Guldemon, T.A. and Hurink, J.L. and Paulus, J.J. and Schutten, J.M.J., Time-Constrained Project Scheduling, *Journal of Scheduling*, 11, 137-148, (2008)
5. S. Hartmann, Packing problems and project scheduling models: An integrative perspective, *Journal of the Operational Research Society*, 51(9), 10831092 (2000)
6. K. Hess and R. Kolisch, Efficient methods for scheduling make-to-order assemblies under resource, assembly area and part availability constraints, *International Journal of Production Research*, 38(1), 207228 (2000)
7. R. Kolisch, Integrated scheduling, assembly area- and part-assignment for large scale make-to-order assemblies, *International Journal of Production Economics*, 64, 127141 (2000)
8. R.H. Moring, Minimizing costs of resource requirements in project networks subject to a fixed completion time, *Operations Research*, 32(1), 89120 (1984)
9. K. Neumann and C. Schwindt, Project scheduling with inventory constraints, *Mathematical Methods of Operations Research*, 56(3), 513533 (2002)
10. K. Neumann, C. Schwindt, and J. Zimmermann, Project Scheduling with Time Windows and Scarce Resources, *Lecture notes in economics and mathematical systems*, Vol. 508, Springer (2002)

On the Effective Approach of Teaching Activity Scheduling in Project Management

Edward Y.H. Lin

Abstract

Project management skills have been widely adopted by the industries of today. The effectiveness of project management techniques accompanied with efficient computing software has prompted further and continuous research efforts in many areas of project management. As a result, project management has become a popular course in the studies of engineering, management, and many other disciplines.

It can be said that the foundation of project management lies on the scheduling of activities in a project. For pathological purposes, we introduce four different approaches of teaching project scheduling to the students with various backgrounds. The first approach is by using the Gantt chart which is easier to be understood by the students in management, particularly the MBA students. The second and the third approaches conduct the scheduling through, respectively, the AOA (Activity On Arc) and AON (Activity On Node) project networks. While each of these two approaches has pros and cons, they are useful to science and engineering students. The last approach is by using the “Activity Adjacency Matrix”. This approach is most helpful in computer coding. It is, therefore, very useful to the students majoring in computer science or information technology.

Our teaching experience shows that, when adopting the proper teaching approach, the student will find project scheduling easier to comprehend and, hence, their interest in the learning of project management is enhanced accordingly.

An Empirical Study of Impacts of Production Mix Efficiency, Product Route Efficiency on Operations Performance and Profitability: A Reverse Logistics Approach.

Kelly O. Weeks • Hongman Gao • Bahram Alidaee • Dharam Rana

1 Introduction

This research studies the impact of two reverse logistics business strategies on profitability of the firm through operations management. The study is employed on scrap steel industry. The first strategy is Production Mix Efficiency (PME), which is involved in the process of producing goods. The second strategy that this research analyzes is Product Route Efficiency (PRE), which engages in the transportation of goods. Finding indicate that operations management alone does not have a positive impact on profitability. However, the two strategies have a positive effect on profitability, which provides a potential answer to firms trying to improve profitability.

Kelly Weeks
Texas A&M University at Galveston
E-mail: weeksk@tamug.edu

Hongman Gao
Jackson State University
E-mail: hongman.gao@jsums.edu

Bahram Alidaee
University of Mississippi
Email: Balidaee@bus.olemiss.edu

Dharam Rana
Jackson State University
dsrana@jsums.edu

2 Manifest Variables

Production Mix Efficiency flexibility is the ability to produce a wide range of products, to accommodate modifications to existing products, and to assimilate new products, all with minimal degradation of performance (Slack, 1987). These “mixes” are based upon factors such as current demand, material availability, equipment, and profit margins. Also, firms may be limited in the amount they can produce and may not be able to match demand. Letmathe and Balakrishnan's (2005) study shows a relationship between optimal Production Mix Efficiency and environmental factors such as waste. This provides evidence that one factor may affect another and that the environmental factor must be taken into consideration when dealing with the Production Mix Efficiency. It is important with regard to profitability to make the most of the materials a firm has. This is achieved by using the materials the firm has to produce a mix of products to achieve maximum profitability, Wahab (2005). Hence, the importance of PME to a firm's profitability and the inclusion of it in this study.

Logistic systems management in distribution channels is a complex process. Integration is needed on all levels to ensure channel efficiency and maintain high levels of customer service (Haas, Murphy, and Lancioni, 2003). These routes can determine whether shipments are on time or late. They also can affect profitability by potentially optimizing route schedules. Information support is also important in the area of product transportation and routing. Authorizing, tracking, and handling returns can positively impact both economic and service quality-related performance (Daugherty, Richey, Genchev, and Chen, 2005). Therefore, Production Route Efficiency must also be considered in this study to ascertain its effect upon profitability.

3 Reverse Logistics Importance

Dowlatshahi (2000) indicates that “from design through manufacture to consumer, firms should explore and integrate reverse logistics as a viable business option in the product life cycle.” Dowlatshahi (2000, 2005) also placed emphasis on the needs for more research in strategic issues of reverse logistics. This research focuses on the reverse aspect of the businesses studied, as research suggests its importance.

The focus of this research was to examine the relationship between Operations Management (OM) and Profitability (PF) from a reverse logistics standpoint using the scrap steel industry as a case study. In addition, this study analyzed some factors (production mix efficiency and product route efficiency) that will affect profitability. Structural equation modeling was used together with confirmatory factor analysis (CFA) to analyze and evaluate the proposed relationships. The conceptual model will be composed of four first-order constructs and six second-order constructs. There are a total of 48 indicator items for the complete model. These 48 items when combined give an accurate overall analysis of the hypotheses, which will be listed in detail.

4 Conclusions

Five hypotheses were developed. Three of the four hypotheses in this research were supported by the data. Hypothesis 1 stated that increased levels of Operations Management (OM) factors will have a significant, non-positive increase on Profitability. Support was found for hypothesis one. Hypothesis 2 states increased levels of Operations Management factors will have a significant, non-positive increase on Production Mix Efficiency. Hypothesis 3 states that increased factors of Production

Mix Efficiency will lead to an increase in Profitability. Support was not found for either Hypothesis 2 or Hypothesis 3. Hypothesis 4 states the following increased levels of Operation Management (OM) factors will have a significant, non-positive increase on Production Route Efficiency. Hypothesis 5 says that as Production Route Efficiency factors increase so to will Profitability. Support was found for Hypotheses 4, and 5.

The results of this research indicate that there is not a direct, positive relationship between operations management and profitability. Therefore strategies such as the two proposed must be implemented in order to achieve the goal of profitability.

References

- Daugherty, P. J., Richey, R.G., Genchev, S.E., and Chen, H., 2005. Reverse logistics: superior performance through focused resource commitments to information technology. *Transportation Research. Part E, Logistics and Transportation Review*, 41E (2), 77-92.
- Dowlatsahi, S., 2000. Developing a theory of reverse logistics. *Interfaces*, 30 (3), 143-155.
- Dowlatsahi, S., 2005. A strategic framework for the design and implementation of remanufacturing operations in reverse logistics. *International Journal of Production Research*, 43 (16), 3455–3480.
- Haas, D. A., Murphy, F.H., and Lancioni, R.A., 2003. Managing reverse logistics channels with data envelopment analysis. *Transportation Journal*, 42(3), 59-69.
- Letmathe, P., and Balakrishnan. N., 2005. Environmental considerations on the optimal product mix. *European Journal of Operational Research*, 167(2), 398-412.
- Slack, N., 1987. The flexibility of manufacturing systems. *International Journal Operations and Production Management*, 7, 35-45.
- Wahab, M. I. W., 2005. Measuring machine and product mix flexibilities of a manufacturing system. *International Journal of Production Research*, 43(18), 3733-3786.

Planning Heuristics for Efficient Data Movement on the Grid

Michal Zerola · Roman Barták · Jérôme
Lauret · Michal Šumbera

1 Problem area

Computationally challenging experiments such as the STAR at RHIC (Relativistic Heavy Ion Collider [1]) located at the Brookhaven National Laboratory (USA), have developed a distributed computing approach (Grid) to face their massive computational and storage requirements. Since the peta-bytes of data produced by STAR are geographically spread, it is necessary to face the question of efficient data transfers and placements in order to bring requested dataset to a particular site for further analyses. Our aim is to create a plan how to transfer data from data-warehouses to the requested site in the shortest time. The presented approach is based on Constraint Programming with initial modeling idea from [2] further extended in [3]. This paper brings a new search heuristic used during the selection of routes.

2 Formal model

The input of the problem consists of two parts. The first part represents the Grid network and file origins. The network, formally a directed weighted graph, consists of a set of nodes $\mathbf{N}$ (sites) and a set of directed edges $\mathbf{E}$ (links). The weight of an edge describes the number of time units needed to transfer one size unit. Information about files' origins is a mapping of each file to a set of sites where the file is available. The second part of the input is a user request, namely the set of files that need to be transferred to a common destination site. The solving process is composed of two stages: a transfer path for each file, i.e., one origin and a valid path from the origin to the destination, is selected (*planning*); second, for each file and its selected transfer path, the particular transfers via links are scheduled in time such that the resulting plan has minimal makespan (*scheduling*). The goal of the scheduling stage is to evaluate the path configuration in the sense of the required makespan. Essentially, it works as an objective function because the realization of the schedule will not depend on particular transfer times calculated in this phase (the measurements show that the

Michal Zerola (corresponding author)
Nuclear Physics Institute, Academy of Sciences, Czech Republic
E-mail: michal.zerola@ujf.cas.cz

difference to the computed makespan is not significant). Both stages iterate until the plan of transfers with the minimal makespan is found (see Alg. 1). In [3] we identified that about 90% of overall time is spent in the planning stage hence we put our effort to improve this stage. The following formalism is used to define a constraint model

Algorithm 1 Pseudocode for a search procedure.

```

makespan  $\leftarrow$  sup
plan  $\leftarrow$  Planner.getFirstPlan()
while plan  $\neq$  null do
  schedule  $\leftarrow$  Scheduler.getSchedule(plan, makespan) {B-a-B on makespan}
  if schedule.getMakespan() < makespan then
    makespan  $\leftarrow$  schedule.getMakespan() {better schedule found}
  end if
  Planner.getNextPlan(makespan) {next feasible plan with cut constraint}
end while

```

describing the planning sub-problem. The set **OUT**(n) consists of all edges leaving node n , the set **IN**(n) of all edges leading to node n . Input received from a user is a set of demands **D** needed at the destination site *dest*. For every demand $d \in D$ we have a set of sources **orig**(d) - sites where the demanded file d is already available. We will present the *link-based* approach for modeling planning constraints, another approach called *path-based* can be found in [3].

The essential idea of the link-based approach is using one decision $\{0, 1\}$ variable X_{de} for each demand and link of the network, denoting whether demand d is routed over edge e or not. Constraints (1-3), ensure that if all decision variables have assigned values then the resulting configuration contains transfer paths. These constraints alone allow isolated loops along with the valid paths and therefore *precedence constraints* (4) are used to eliminate such loops.

$$\forall d \in \mathbf{D} : \sum_{e \in \mathbf{OUT}(n) | n \in \mathbf{orig}(d)} X_{de} = 1, \quad \sum_{e \in \mathbf{IN}(n) | n \in \mathbf{orig}(d)} X_{de} = 0 \quad (1)$$

$$\forall d \in \mathbf{D} : \sum_{e \in \mathbf{OUT}(dest(d))} X_{de} = 0, \quad \sum_{e \in \mathbf{IN}(dest(d))} X_{de} = 1 \quad (2)$$

$$\forall d \in \mathbf{D}, \forall n \notin \mathbf{orig}(d) \cup \{dest(d)\} : \\ \sum_{e \in \mathbf{OUT}(n)} X_{de} \leq 1, \quad \sum_{e \in \mathbf{IN}(n)} X_{de} \leq 1, \quad \sum_{e \in \mathbf{OUT}(n)} X_{de} = \sum_{e \in \mathbf{IN}(n)} X_{de} \quad (3)$$

Precedence constraints (4) use non-decision positive integer variables P_{de} representing possible start times of transfer for demand d over edge e . Let dur_{de} be the constant duration of transfer of d over edge e . Then constraint

$$\forall d \in \mathbf{D} \forall n \in \mathbf{N} : \sum_{e \in \mathbf{IN}(n)} X_{de} \cdot (P_{de} + dur_{de}) \leq \sum_{e \in \mathbf{OUT}(n)} X_{de} \cdot P_{de} \quad (4)$$

ensures a correct order between transfers for every demand, thus restricting loops. Unfortunately, constraints (4) do not restrict the domains of P_{de} until the values X_{de} are known and therefore we suggest using a redundant constraint (5) to estimate better

the lower bound for each P_{de} . Let $start$ be the start vertex of e not containing demand d ($start \notin orig(d)$):

$$\min_{f \in \mathbf{IN}(start)} (P_{df} + dur_{df}) \leq P_{de} \quad (5)$$

Variables P_{de} can be used not only to break cycles but also to estimate makespan of the plan. The idea is that according to the number of currently assigned demands per some link and their possible starting times, we can determine the lower bound of the makespan for schedule that will be computed later in the scheduling stage. Hence if we have some upper bound for the makespan (typically obtained as the best solution from the previous iteration of planning and scheduling) we can restrict plans in next iterations by the following constraint:

$$\forall e \in \mathbf{E} : \min_{d \in \mathbf{D}} (P_{de}) + \sum_{d \in \mathbf{D}} X_{de} \cdot dur_{de} + SP_e < makespan, \quad (6)$$

where SP_e stands for the value of the shortest path from the ending site of e to $dest$.

3 Search heuristics

The constraint model needs to be accompanied by a clever branching strategy to achieve good runtimes. In [3] we proposed *FastestLink* variable selection heuristic that suggests for each demand d to instantiate first the variable X_{de_j} corresponding to the fastest link ($j = \arg \min_{i=1, \dots, m} dur_{de_i}$). In this paper we propose a new variable selection heuristic that exploits better the actual transfer times by using information from variables P_{de} . In particular, the heuristic, called *MinPath*, suggests to instantiate first variable X_{de} such that the following value is minimal:

$$\inf P_{de} + dur_{de} + SP_e, \quad (7)$$

where $\inf P_{de}$ means the smallest value in the current domain of P_{de} .

4 Experiments

We compared the performance of the *FastestLink* and *MinPath* heuristics and a Peer-2-Peer model [4] that is currently the most frequently used approach to solve the problem. The constraint model was implemented in the Java-based constraint programming library **Choco** 2.0.0.3 and all experiments were performed on Intel Core2 Duo (@1.6GHz) with 2GB of RAM, running Debian GNU Linux operating system. We used a realistic-like network graph and file distribution as described in [3]. Figure 1 shows that convergence of the new *MinPath* heuristic is faster than the *FastestLink* and both heuristics achieve better makespan than the P2P approach. Table 1 shows similar comparison of heuristics and the P2P model including the time when the best solution was found for several input instances.

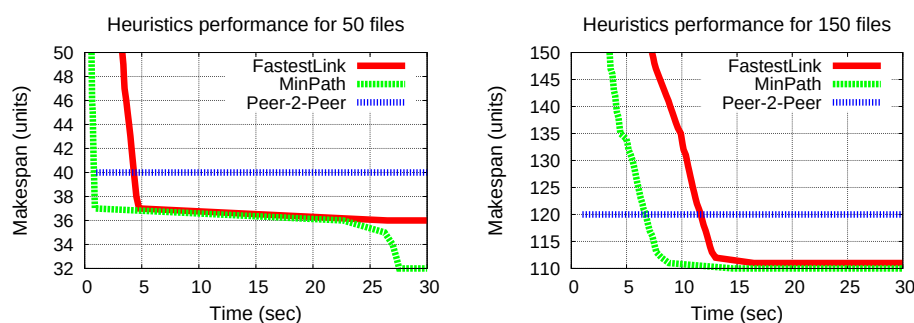


Fig. 1 Convergence of makespan during the search process for *FastestLink* and *MinPath*.

Files	Solution time		Makespan		
	<i>FastestLink</i>	<i>MinPath</i>	<i>FastestLink</i>	<i>MinPath</i>	<i>P2P</i>
25	3.862	1.431	14	14	24
50	26.508	27.556	36	32	40
100	8.627	3.176	73	73	80
150	16.52	14.618	111	110	120
200	26.167	14.031	146	146	160

Table 1 Comparison of heuristics with emphasis on time when the best solution was found and the makespan.

5 Conclusions

In this paper we tackle the complex problem of efficient data movements on the network within a Grid environment. The problem itself arises from the real-life needs of the running nuclear physics experiment STAR and its peta-scale requirements for data storage and computational power as well. We have focused on improvements of the search heuristic for the planning stage using domains of start time variables that were added to the constraint model with precedence constraints. Experimental results confirmed that the new heuristic converges faster toward the optimal solution.

Acknowledgements The investigations have been partially supported by the IRP AVOZ 10480505, by the Grant Agency of the Czech Republic under Contract No. 202/07/0079 and 201/07/0205, by the grant LC07048 of the Ministry of Education of the Czech Republic and by the U.S. Department Of Energy.

References

1. J. Adams, e.a.: Experimental and theoretical challenges in the search for the quark gluon plasma: The STAR collaboration's critical assessment of the evidence from RHIC collisions. *Nuclear Physics A* **757**, 102–183 (2005)
2. Simonis, H.: Constraint applications in networks. In: F. Rossi, P. van Beek, T. Walsh (eds.) *Handbook of Constraint Programming*, chap. 25, pp. 875–903. Elsevier (2006)
3. Zerola, M., Barták, R., Lauret, J., Šumbera, M.: Using Constraint Programming to Plan Efficient Data Movement on the Grid. (Submitted to ICAPS 2009) (2009)
4. Kaashoek, M.F., Stoica, I. (eds.): *Peer-to-Peer Systems II*, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22, 2003, Revised Papers, *Lecture Notes in Computer Science*, vol. 2735. Springer (2003)

Order acceptance and scheduling in a single-machine environment: exact and heuristic algorithms

Fabrice Talla Nobibon · Jade Herbots · Roel
Leus

1 Introduction

The integration of project selection and workload planning has been shown to be essential for overcoming the conflicting objectives between the sales department and the production department of a firm [2].

This paper studies order acceptance and scheduling in a single-machine environment in the case where a pool consisting of firm planned orders F as well as potential orders $\bar{F}$ is available from which an over-demanded company can select; all jobs can be executed from the beginning of the planning period. There are two decisions to be made: which jobs in $\bar{F}$ to accept, and in which order to process the selected subset. The capacity available for processing the accepted orders is limited and each order i is characterized by a known processing time p_i , delivery date d_i , revenue Q_i and the weight w_i representing a penalty per unit-time delay beyond d_i in the delivery to the customer. The aim is to maximize the total net profit, that is, the sum of the revenues of the selected jobs minus applicable total weighted tardiness penalties incurred for those jobs. We establish a number of theoretical results and present two mixed-integer linear formulations of the problem. We devise exact and heuristic algorithms and report results on the comparison of the different algorithms proposed based on computational experiments.

2 Theoretical results and linear formulations

The order acceptance and scheduling problem on a single machine environment is strongly NP-Hard, since it contains the scheduling problem with total weighted tardi-

Fabrice Talla Nobibon
Katholieke Universiteit Leuven
E-mail: Fabrice.TallaNobibon@econ.kuleuven.be

Jade Herbots
Katholieke Universiteit Leuven
E-mail: Jade.Herbots@econ.kuleuven.be

Roel Leus
Katholieke Universiteit Leuven
E-mail: Roel.Leus@econ.kuleuven.be

ness penalties, which is known to be strongly NP-Hard [3]. Moreover, we prove that unless $P = NP$, it is impossible to derive a polynomial time algorithm that guarantees a constant factor approximation.

Let $y_i \in \{0, 1\}$ be the binary decision variable equal to 1 if the job i is accepted and 0 otherwise. The binary variable $x_{it} \in \{0, 1\}$ is used to identify the position of each accepted job i when processed; x_{it} equals 1 if job i is accepted and is the t^{th} job to be processed and 0 otherwise. The binary variable z_{ji} ($i \neq j$) is equal to 1 if both jobs i and j are accepted and job j is executed before job i ; otherwise z_{ji} takes the value 0. Finally, a real variable T_i represents the tardiness of job $i = 1, \dots, n$. A mixed-integer linear formulation for the order acceptance and scheduling problem can be expressed as:

$$(MIP) \quad \text{maximize} \quad \sum_{i=1}^n Q_i y_i - \sum_{i=1}^n w_i T_i \quad (1)$$

$$\text{subject to} \quad y_i = 1, \quad \forall i \in F, \quad (2)$$

$$y_i = \sum_{t=1}^n x_{it}, \quad i = 1, \dots, n, \quad (3)$$

$$\sum_{i=1}^n x_{it} \leq 1, \quad t = 1, \dots, n, \quad (4)$$

$$z_{ji} \leq y_i, \quad z_{ji} \leq y_j, \quad i, j = 1, \dots, n, \quad i \neq j, \quad (5)$$

$$\sum_{q < t} x_{jq} + \sum_{q \geq t} x_{iq} \leq 1 + z_{ji}, \quad t = 2, \dots, n, \quad (6)$$

$$T_i \geq \sum_{j=1}^n p_j z_{ji} + p_i y_i - d_i, \quad i = 1, \dots, n, \quad (7)$$

$$y_i, x_{it}, z_{ji} \in \{0, 1\}, \quad T_i \geq 0, \quad i, j, t = 1, \dots, n. \quad (8)$$

We prove that a solution to (MIP) can be obtained by solving a slightly modified version where the integrality constraints on the variables y_i and z_{ji} are relaxed. The formulation can be further strengthened by adding the following set of cuts: $\sum_{i=1}^n x_{it+1} \leq \sum_{i=1}^n x_{it}$, $t = 1, \dots, n-1$.

The second formulation is a time-indexed formulation [1]. It is based on a discretization of the time horizon into T periods, (in our case $T = \sum_{i=1}^n p_i$) and uses the binary variable $x_{it} \in \{0, 1\}$, $i = 1, \dots, n$, $t = 1, \dots, T$, that equals 1 if job i is selected and its execution starts during period t and 0 otherwise, where the period t begins at time $t-1$ and ends at time t . Computational results show that the latter formulation is suited for problem instances with medium or small processing times (less than 20) while we recommend the use of the former formulation for problem instances with larger processing times.

3 Implicit-enumeration algorithms

We present two branch-and-bound (B&B) algorithms to perform implicit enumeration of the solution space. These B&B algorithms are inspired by the work of Slotnick and

Morton [7]. The first B&B algorithm (subsequently referred to as two-phase B&B) is a hierarchical algorithm that performs selection and scheduling separately. At each node of the branching tree, it uses the B&B algorithm for sequencing with total weighted tardiness developed by Potts and Van Wassenhove [5] to schedule the selected set of jobs. The second algorithm (called direct B&B) performs both selection and scheduling simultaneously. Some tools used by Potts and Van Wassenhove [5] to improve the running time of their B&B are adapted for the direct B&B, namely the precedence graph and the adjacent job interchange rule.

4 Heuristics

We present six heuristics for solving the problem. These heuristics are developed based on the structure of the problem and the exact algorithms presented above. The first is a slightly modified version of the myopic heuristic as described by Rom and Slotnick [6]. The modification allows to take into account the jobs in F . The second is an improvement of the first heuristic. Here, the B&B developed by Potts and Van Wassenhove [5] is used to replace the Rachamadugu and Morton heuristic [4] in the myopic algorithm. The third heuristic is based on the LP relaxation of the first linear formulation MIP while the fourth is based on the LP relaxation of the time-indexed formulation. The fifth heuristic is a depth-first B&B heuristic based on the two-phase B&B algorithm while the last one is a truncated direct B&B heuristic.

5 Computational experiments

During the presentation, we will discuss results on the comparison of the efficiency of the exact and heuristic algorithms developed in this paper based on computational experiments, including the solution of the linear formulations using CPLEX.

References

1. Bigras, L.-P., Gamache, M. and Savard, G.: Time-indexed formulations and the total weighted tardiness problem. *INFORMS Journal on Computing*, 20: 133-142 (2008)
2. Guerrero, H.H. and Kern, G.M.: How to more effectively accept and refuse orders. *Production and Inventory Management Journal*, 4: 59-62 (1998)
3. Lenstra, J.K., Rinnooy Kan, A.H. and Brucker, P.: Complexity of machine scheduling problems. *Annals of discrete mathematics*, 1: 343-362 (1977)
4. Morton, T.E and Rachamadugu, R.M.: Myopic heuristics for the single machine weighted tardiness problem. Working Paper No. 30-82-83, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA (1982)
5. Potts, C.N. and Van Wassenhove, L.N.: A branch and bound algorithm for the total weighted tardiness problem. *Operations research*, 33: 363-377 (1985)
6. Rom, W.O. and Slotnick, S.A.: Order acceptance using genetic algorithms. *Computers & Operations Research*, 36: 1758-1767 (2009)
7. Slotnick, S.A. and Morton, T.E.: Order acceptance with weighted tardiness. *Computers & Operations Research*, 34: 3029-3042 (2007)

Single machine re-entrant jobs scheduling in the car factory paint shop

Grzegorz Pawlak · Marek Rucinski

1 Introduction

The problem of increasing productivity is always important in the car factory. In the paper the problem of scheduling cars in the car factory paint shop has been considered. The particular problem is modeled and simple algorithms, for the flow control of the cars through the production line of the paint shop station have been proposed.

The general objective is to schedule cars in order to increase the throughput rate through the painting station on the base color painting machine stage. The throughput rate depends on the size of the car blocks of the same colors and the car types in the car sequence constructed by the production plan. The input cars sequence is optimized in the input buffer where the car bodies are sorted in order to minimize the color switchings in the painting machine stage. Then cars are painted in the painting station and then go to the inspection station where the quality of the painting is checked. Some of the cars are redirected back to the input sequence for repainting. The previous input sequence is disturbed by the reentrant cars and the number of color changes in the painting station increases. According to the technological process the repainting of the same car could be done several times. Some of the cars are obligatory repainted, covered by the double base color layers, due to technological requirements. The reentrant line consists of the limited buffer where the cars could be sorted. The repainting of the car must be done in the time window, when the introduction to the original car sequence has to be performed. That limitation in the processing must avoid deadlock in the process of painting line.

The Figure 1 shows the production area and the process chart. The M represents the base color painting machine; B represents limited reentrant line buffer; IS states the *Inspection Stage* where the car routing is decided if car goes to the output line or back to the repaint input line.

Grzegorz Pawlak
Poznan University of Technology, Institute of Computing Science
E-mail: grzegorz.pawlak@cs.put.poznan.pl

Marek Rucinski
Poznan University of Technology, Institute of Computing Science
E-mail: mrucinski@skno.cs.put.poznan.pl

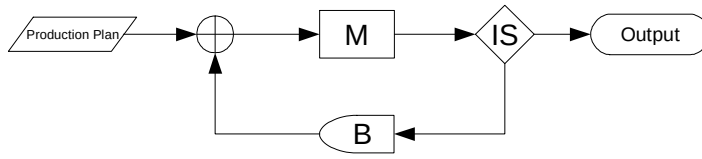


Fig. 1 The production area chart

2 The problem formulation

The problem has been formulated as the single machine scheduling problem with reentrant jobs and set-up times $J_j = (T_j^1, T_j^2, \dots, T_j^{n_j})$ and $n_j \geq 1$ where n_j is the number of painting operations for each car. Each job has addition two properties $type_j$ representing the type of the car and the $color_j$ - the color of the car. The processing time p_j is represented by the time necessary to paint the car of the particular type. Between the subsequent cars the set-up time is introduced. There are two types of the set-ups. First s_i introduced when two subsequent cars have the same color and second s_j when two subsequent cars have different colors. The set-up time is connected with the cleaning process of the painting guns. The set-ups holds inequality $0 \leq s_i < s_j$. On the re-painting line the buffer B is located with finite capacity N_B . The cars must leave buffer B within the *Time Window* $T_j^i : (r_{T_j^i}, d_{T_j^i})$ where $r_{T_j^i}$ is the ready time of the car to be repainted and $d_{T_j^i}$ is the deadline when the car must be introduced into the input sequence of the cars and painted at the painting station. The input car sequence is predefined by the production plan. The criterion is C_{max} . It is relevant because of the fixed speed of the line. Then the C_{max} criterion is equivalent to the throughput rate of the production in the defined time horizon. In addition the number of the external set-up times has been also measured.

3 Complexity of the problem

The deterministic version of the problem with n_j known in advance is in the general case NP-hard. One can easy transform NP-hard scheduling problem denoted as $1|chain(l)|C_{max}$, described in [1], into the described above. The presented problem is even more complex by introducing the set-up times, time windows and finite capacity buffer. Moreover, in the real case the value of n_j is not known in advance. Then the on-line approach could be applied.

4 Algorithms and results

In the paper two on-line adaptive algorithms were presented. The algorithms were tested on the real data. The data were collected from the car factory in the one month

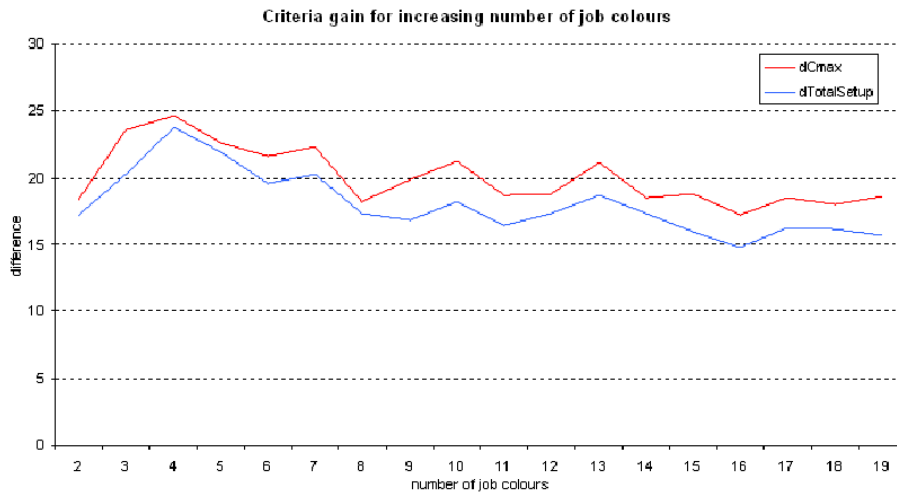


Fig. 2 Gain for applying the proposed algorithm in comparison with the existing control algorithm

time period. The algorithm were compared to the existing algorithm controlling the process. The original one were generally based on the FIFO rule. The proposed algorithms utilized the different dispatching rule. In the given time window the cars were sorted in the buffer and introduced to the main sequence maximizing the car block size of the same color. Some results were presented in the Figure 2. The gain in comparison to the number of job colors. Introducing the proposed algorithms can gain for increasing number of the job colors from 15 to 25.

5 Conclusions

In the paper the practical approach for the solving the real car scheduling problem were proposed. The car painting problem was modeled as the one machine scheduling problem with the reentrant jobs with additional constraints. The proposed adaptive on-line algorithms were tested on the real data and their effectiveness was showed. Further research will consider meta heuristic approach for solving the problem more accurately. The correlation between the block size for cars of the same color, in the input sequence and the distribution of the repainted cars for the particular color should be evaluated.

Acknowledgements Research supported by Grant No. N519188933 Polish Ministry of Science and Higher Education

References

1. E.D. Wikum, D.C. Llewellyn, and G.L. Nemhauser, *One-machine generalized precedence constrained scheduling problems* Operations Research Letters, 16(2), 1994, pp. 87-99.

Real World Optimisation through Spin Out Activity

B. McCollum, R.S.K. Kwan, P.J. McMullan

1 Introduction

A key research issue in Scheduling and Operational Research is the goal of closing the gap between theory and practice [1]. All too often researchers content themselves with working on abstractions of problems [2] which can make it difficult to implement actual solutions for real world problems. Within the fields of optimization and scheduling, heuristic approaches are used extensively in helping provide solutions. It is often the case that tuning of these approaches to individual problem domains, representations and indeed datasets are required in the search and provision of better solutions. Hyperheuristics [3] have recently been introduced as a concept to encompass the development of more general techniques which have applicability across different problem domains. More research initiative is required to aid the applicability of leading edge research within industrially defined real world problems. The approach taken here is to describe and discuss two cases where solutions have been provided to real world problems. EventMAP Limited and Tracsis plc are two spin out companies which provide solutions to real scheduling problems where optimization is a central issue. This paper describes the different approaches taken by the Companies, which reflect the underlying differences in the problems to be solved. The aim is to understand how the linkages between research based techniques and spin out activity can help solve other key real world industrial optimization problems [4,5]. At the conference, both approaches will be discussed highlighting experiences, similarities and differences.

B G C McCollum

School of Electronics, Electrical Engineering and Computer Science, Queen's University,
Belfast

E-mail: b.mccollum@qub.ac.uk

R.S.K Kwan

School of Computing, University of Leeds

E-mail: r.s.kwan@leeds.co.uk

P J McMullan

School of Electronics, Electrical Engineering and Computer Science, Queen's University,
Belfast

E-mail: p.j.mcmullan@qub.ac.uk

2 The EventMAP Approach

Timetabling of EventMAP Limited provides software to the university sector for the purpose of timetabling and managing resource. The Company currently has two offerings, OptimeExam and OptimeEvent, to deal with different institutional timetabling scenarios. While OptimeExam helps with the production of the examination timetabling, OptimeEvent helps timetable all aspects of an institution's curriculum.

Timetabling represents an organisational task which requires the interaction of humans and technology. To varying extents, individuals keep data issues updated and ensure individual and institutional preferences are taken care of. The software aides the process by allowing electronic representation of the timetabling issues within the institution. This is achieved by providing an intuitive interface through which timetable scenarios can be modeled. This modeling process takes place in terms of events to be placed, the availability and preferential use of time and resource and student qualitative considerations. Importantly, once a representation of the problem is achieved, the software, powered by the EventMAP Scheduling Engine, is capable of providing high quality solutions through the employment of research led search methodologies.

Solutions are produced in a two phased process [7,8]. Initially the problem is modeled as a graph coloring problem allowing events to be ordered in a predefined list before being placed in a timetable structure, i.e. times and rooms. The ordering of events is determined by an adaptive heuristic ordering. At the end of this phase the feasible solution e.g. one that satisfies all hard constraints obtained, is improved by remodeling the problem as one of search and the associated application of metaheuristic techniques e.g. extended great deluge. Other search based techniques are maintained and used as required. An overview of achievable results will be presented. Evaluation of the resultant solution is driven by flexibility for students and staff and efficiency of resources e.g. space. This is therefore a combination of qualitative and quantitative issues and must be defined for each institution. Indeed, prior modeling is often used to help establish the amount of potential flexibility available to the provision of an institution wide solution.

Powered by the EventMAP Scheduling Engine, the software is able to provide high quality solutions for students in terms of spread throughout the designated timetabling session. It is clear that issues associated with examination and centralized course timetabling must be understood and represented before optimisation techniques can produce effective solutions [2]. The provision of course timetabling solutions to large decentralised institutions requires a combination of leading edge optimization within an effective information management software product. EventMAP have teamed up with CELCAT® in the development of such a solution. Central to the collaboration is addressing the gap which currently exists between research and practice in timetabling research. Various areas have been outlined [5] and progress will be discussed.

3 The Tracsis Approach

Tracsis plc provides an automatic crew scheduling software called TrainTRACS [6] for use by planners within train operating companies. The software is also used by train franchise bid teams in major franchise tendering exercises.

Crew scheduling is mission critical in operating passenger rail services. Validity of solutions is therefore of utmost importance. Nearly right is simply not acceptable. However, the problem is made difficult by a very wide variety of complex rules, constraints, and local knowledge, which are hard to capture and model fully and accurately.

For franchise bidding, the solutions do not have to be proven to be operable at the detailed level. However, the consequences of making a wrong estimate of crewing levels required could be dire, particularly so if the bidder won the franchise but is not able to implement what has been planned and costed. It is also important for the bidders to evaluate

many alternative plans before going for the best one. However, the timescale of the bidding process is extremely tight and that calls for an automatic crew scheduling software.

The Tracsis approach works on a comprehensive model of input data and parameters such that virtually all problem scenarios from different train operating companies can be accommodated. This has been achieved incrementally over many years directly collaborating with the train companies. The Generate and Select approach [9] employed helps: hundreds of thousands of legal potential crew shifts are first generated by heuristics; then the cheapest subset, of say a few hundred shifts, is selected to form the solution schedule. Since in the Generation Phase there is no consideration of how individual potential shifts would fit in with each other, so long as they individually have satisfied all the local rules and constraints, heuristics can be added relatively easily to cater for new scheduling conditions. Powerful optimization is achieved in the Selection Phase, which essentially can be solved generically as a set covering integer linear programme.

There are only two main objectives, which are the minimization of the total number of shifts and the minimization of the total wage costs. An objective function can be formulated easily as a weighted sum of these two aspects to be minimized. With respect to the stakeholders, mainly the management and the planners, they normally would not be in conflict in terms of the crew scheduling objectives.

The capability of TrainTRACS reflects the accumulation of a tremendous amount of research over time in association with the University of Leeds. However, the research has been not just on pure theoretical grounds, but has always ensured that real life scenarios and data are used. Through the University's commercial arm in early years and through Tracsis since the spin-out was founded in 2004, vital industrial links are fostered. The collaboration with a train company usually starts in a project piloting the use of TrainTRACS. The industry has to be convinced of optimization software by solving their problems using their own real data. Often, new scheduling scenarios and conditions would arise in these pilot trials. They are good tests on the versatility of the optimization system. Most of the time, existing system features could cope even though they might have been designed for other scenarios and conditions originally. In other cases, new features could be added to the Generation Phase quite quickly as explained above. And after the pilot trials from time to time, any temporary or new fixes would be incorporated into the system model in an incremental manner. As train companies have become established users of TrainTRACS, new problem scenarios and scheduling conditions would be fed back as support requests to Tracsis which informs further research.

4 Conclusion

This work is motivated by the need to close the gap between research and practice which currently exists in many areas within the areas of optimization and planning. The transfer of the achievements and skill developed within research to industrial real world problems is a central knowledge transfer issue. The situation is complicated by two major issues i.e. over simplification of problem representation and the fragility of the search techniques used. Heuristic based techniques, although providing good results in many cases, are fragile in the sense that their setup is often over dependent on the problem domain, representation and dataset investigated.

Both EventMAP and Tracsis represent companies which are linking research and practice within the area of optimization and scheduling. It is proposed through analyzing and understanding differences between the associated approaches, much light will be shed on the requirements of techniques to help with real world scenarios. While EventMAP deal with many stakeholders with many, often conflicting, objectives, the nature of the problem faced by Tracsis is closer to a hardcore combinatorial optimization problem where near absolute precision in modeling is required. In general stakeholders are in unison in their objectives. Another major difference is the style of implementation of the associated approaches. While the EventMAP approach often requires a high degree of customization, particularly for large institutions, Tracsis implementations are largely generic. The techniques,

similarities and differences will be discussed in helping to understand the nature between research and practice.

References

1. Review of Research Status of Operational Research in the UK, 2004
2. B. McCollum, A Perspective on Bridging the Gap between Research and Practice in University Timetabling (2007), Practice and Theory of Automated Timetabling VI (eds. E.K.Burke and H.Rudova), Lecture Notes in Computer Science Volume 3867, Springer 2007, pages 3-23
3. E.K.Burke, G.Kendall and E.Soubeiga, A Tabu Search Hyper-Heuristic for Timetabling and Rostering, Journal of Heuristics Vol 9(6), December 2003, pages 451-470
4. B. McCollum, P. McMullan, J. Newall, JP Lane , A workable scheduling algorithm, The 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), Nottingham, August 2003, pages 570-572
5. B. McCollum, P. McMullan, E. Burke, G. Kendall, A. Gough, A. Jones, Academic Timetabling: Linking Research and Practice, Practice and Theory of Automated Timetabling (PATAT 2008), Montreal, 19-22 August 2008
6. Kwan, Raymond S K; Kwan, Ann. Effective search space control for large and/or complex driver scheduling problems. Annals of Operations Research, vol. 155, pages 417-435. 2007.
7. B. McCollum, P. J. McMullan, A. J. Parkes, E.K. Burke, An Extended Great Deluge Approach to the Examination Timetabling Problem, submitted to MISTA, Dublin 09
8. McMullan, P, An Extended Implementation of the Great Deluge Algorithm for Course Timetabling, Lecture Notes in Computer Science, Springer, Vol 4487, pages 538-545, 2007
9. R.S.K Kwan,(2004). "Bus and train driver scheduling." In J. Y.-T. Leung (ed.), Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Chapter 51, pages 1- 20

Scheduling in a micro-biological laboratory: heuristics for sequencing food sample tests, to co-ordinate with production of perishable media materials

Celia Glass • Konstantin Chakhlevitch

1 Abstract

This paper explores the scheduling issues associated with co-ordinating two key processes in a food testing micro-biology laboratory. Most of the microbiology tests in the laboratory are carried out on glass dishes, in an agar compound. The type of agar employed is very specific to the nature of the test. Each food sample requires a whole suite of tests to be performed successfully in order to be certified fit for consumption. Thus the sequence of samples determines parallel demand for a range of agars. A major complication is the limited lifetime over which the agars are effective. The co-ordination of scheduling test samples with production of agars in batches on identical, parallel machines, is the topic of interest.

We focus on sequencing of test samples in such a way as to ensure that agars are consumed as quickly as possible since their usage starts. Therefore, the problem of sequencing test samples reduces to one of sequencing their agar combinations. A sequence of agar combinations can be represented as a $(0,1)$ -matrix which columns correspond to agar types and rows specify agar combinations for test samples. Hence, the problem can be formulated in the following way: given a binary matrix, find a permutation of its rows such that the total number of zero-entries situated between the blocks of consecutive ones in the columns of the permuted matrix is minimized. Such a formulation has other important applications in the design of the efficient storage schemes for sparse matrices, and in the physical mapping of chromosomes in computational biology. The problem is proved to be NP-hard and no effective solution methods are known from the literature. It is, however, related to the Consecutive Ones Problem which is studied in the literature.

In this paper, we present an optimization approach for the problem described above, which is based on indirect enumeration of the solution space. The idea behind the technique is to define classes of solutions by specifying their common structures. Then, an optimal solution within each class is determined and the best of these provides an optimal solution to the original problem. Although this approach is more efficient than complete enumeration of all

Celia A. Glass

Cass Business School, City University London, 106 Bunhill Row, London EC1Y 8TZ, UK
E-mail: C.A.Glass@city.ac.uk

Konstantin Chakhlevitch

Cass Business School, City University London, 106 Bunhill Row, London EC1Y 8TZ, UK
E-mail: Konstantin.Chakhlevitch.1@city.ac.uk

feasible solutions, i.e., all possible permutations of rows of the original matrix, it can only be applied to matrices with a small number of columns. Therefore, we develop two heuristic algorithms which quickly construct feasible solutions to the problems. The first heuristic is a greedy insertion algorithm while the second one is a more complicated technique which constructs a solution matrix in two directions, from the top and the bottom.

We report the results of our algorithms for a wide range of randomly generated problem instances. We show that the first heuristic works well for small instances and the second one is able to produce optimal or near-optimal outcomes for the whole range of problem instances. In addition, we show how our structural insights into the problem enable data from the microbiological laboratory to be represented as relatively small problem instances, which can then be efficiently tackled by our exact algorithm.

A self-organising hyper-heuristic framework

E. Özcan · M. Mısı́r · E. K. Burke

1 Introduction

Hyper-heuristics can be thought of as heuristic management methodologies which are motivated by the goal of building a high level of generality in the scope of computational search methodologies [2] [7]. Most meta-heuristics (and other search methods) perform a search over the solution space directly, whereas, hyper-heuristics as high level strategies search a heuristic space. One of the hyper-heuristic frameworks makes use of multiple perturbative (improvement) heuristics in a single-point iterative search setting. At each step, one of the low level heuristics is selected and applied to the candidate solution at hand. The resulting move is either accepted or rejected based on some problem independent criteria. This study focuses on such *perturbative hyper-heuristics*.

In [5], two types of low level heuristics are identified: *mutational heuristics* and *hill climbers*. A mutational heuristic perturbs a given candidate solution in a systematic way using a stochastic component without any expectation regarding the change in the quality of the solution. On the other hand, a hill climber returns a solution that has either better or equal quality with the input solution. [5] investigates four different types of perturbative hyper-heuristic frameworks that aim to make better use of multiple mutational and hill climbing heuristics as illustrated in Figure 1: F_A , F_B , F_C and F_D . In the type A , all low level heuristics are utilised simultaneously. The type B framework functions similarly. The only difference is that if the heuristic selection

Ender Özcan · Edmund K. Burke
University of Nottingham, School of Computer Science & IT
The Automated Scheduling, Optimisation and Planning (ASAP) Research Group
Jubilee Campus, Wollaton Road, Nottingham
NG8 1BB, UK
E-mail: {exo,ekb}@cs.nott.ac.uk

Mustafa Mısı́r *
Katholieke Universiteit Leuven, Department of Computer Science
Celestijnenlaan 200A
3001 Heverlee-Leuven/BELGIUM
KaHo Sint-Lieven, IT Research Group
Gebroeders Desmetstraat 1
9000 Gent/BELGIUM
E-mail: mustafa.misir@kahosl.be

method chooses and applies a mutational heuristic to the candidate solution, then a predefined hill climber is invoked afterwards. If the chosen heuristic is a hill climber, the second process does not take place. The resulting solution is fed into the move acceptance strategy. In the type C framework, the low level heuristic set contains only mutational heuristics. A predetermined hill climber is applied following the chosen mutational heuristic. The last framework is the most general framework. It uses two hyper-heuristics successively at each iteration. The first hyper-heuristic manages a set of mutational heuristics, while the second one manages a set of hill climbers. Both hyper-heuristics can be the same or each hyper-heuristic mechanism can have its own heuristic selection and move acceptance strategy. It is also possible to design a type D framework which passes information between hyper-heuristics. The experimental results show that F_C is superior to the others. Although F_D has not performed well, this might be due to the choice of hyper-heuristics which function independently and do not communicate any information with each other. Notice that F_C is a special case of F_D .

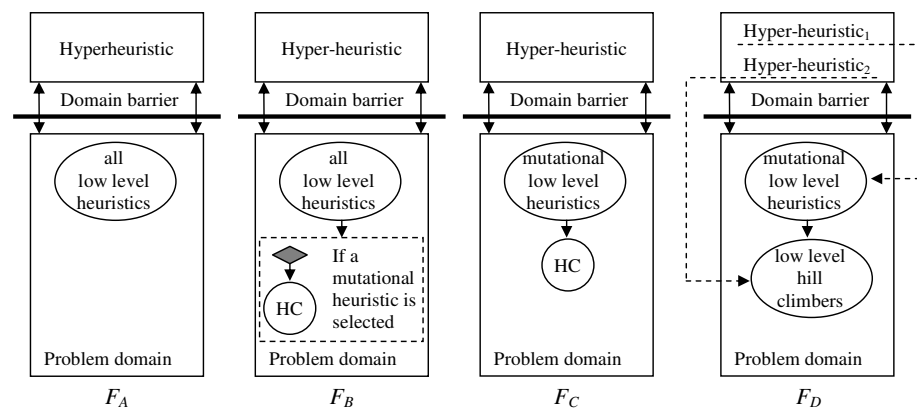


Fig. 1 Hyper-heuristic frameworks presented in [5] that make use of multiple mutational and hill climbing (HC) heuristics.

In this study, a hyper-heuristic framework of type D is investigated. It is assumed that the nature of each low level heuristic is not known. Reinforcement learning is employed as a mechanism to adaptively classify low level heuristics into two categories (mutational heuristics and hill climbers). Moreover, instead of two separate hyper-heuristics, a unified single hyper-heuristic that self organises itself to operate as a type D framework is utilised.

2 A self organising hyper-heuristic framework based on reinforcement learning

Hyper-heuristic research could benefit from exploring how machine learning techniques can perform better choices during the search process [1] [3] [4] [6] [8]. Within the context of perturbative heuristics, these learning techniques are utilised by the heuristic selection component. The most common approach is the reinforcement learning due to its

simple structure and effectiveness. A significant study on reinforcement learning based hyper-heuristics is provided by Nareyek (2004) in [4]. Depending on the performance of a given heuristic, it is either rewarded or punished. This study investigates different rates for reward and punishment as well as different heuristic selection mechanisms using the scores. This study adapts the same idea, but employs a different strategy during the heuristic selection process.

In our approach, we use +1 as a reward factor as suggested in [4]. For the punishment, three different scoring schemes are utilised, denoted as s_i :

$$s_1 : w_i = w_i - 1 \quad (1)$$

$$s_2 : w_i = w_i/2 \quad (2)$$

$$s_3 : w_i = \sqrt{w_i} \quad (3)$$

Given n heuristics, these scores indicate the individual performance of each low-level heuristic. Moreover, they are used to generate two different heuristic partitions from the whole low level heuristic set. These sets are dynamically determined based on the average score of the whole low level heuristics at any given iteration:

$$avr = \sum_{\forall i} w_i / n \quad (4)$$

Heuristics having a score below the average form the set C_1 ($C_1 = \{\forall i; w_i < avr\}$), while heuristics having a score above or equal to the average form the set C_2 ($C_2 = \{\forall i; w_i \geq avr\}$). *Improving or Equal* (IE) is preferred as move acceptance criteria, since it is reported to be the best choice for the benchmark function optimisation in [5]. The heuristic selection methods using the punishment scoring schemes s_1 , s_2 and s_3 are denoted as RL_1 , RL_2 and RL_3 , respectively.

At any point of search, it is expected that the heuristics in C_2 will be hill climbers and the rest in C_1 will be the mutational heuristics. The proposed hyper-heuristic framework works in two consecutive stages by choosing a heuristic from C_1 randomly, applying it to the current solution and feeding the new solution into the move acceptance mechanism in the first stage. Then, the same procedure is followed for the C_2 heuristics in the second stage. We can categorise this hyper-heuristic as a type D framework. Three reinforcement learning based hyper-heuristics of this type are tested over 13 well-known mathematical benchmark functions (see [5] for more details). Each hyper-heuristic denoted as FD_1 , FD_2 and FD_3 utilises different punishment rates as provided in Equations 1, 2 and 3, respectively.

3 Results

During the experiments, Pentium IV 3 GHz LINUX (Fedora Core 8) machines having 2 Gb memories have been used. Fifty runs are performed during each test on a benchmark function. For a fair comparison between all algorithms, the experiments are terminated if the execution time exceeds 600 CPU seconds or the expected fitness (Opt) is achieved. The success rate denotes the ratio of successful runs in which the expected fitness is achieved to the total number of runs. So, for instance, if an algorithm finds the optimum solution for the given benchmark for each trial, then its success rate is 1.0.

The results are compared to the traditional reinforcement learning with maximal utility selection. In Table 1, the ranking of each hyper-heuristic based on the success rates (and in the case of equality based on the number of fitness evaluations) are

Function	Dim	Opt	RL_1	RL_2	RL_3	FD_1	FD_2	FD_3
Sphere	10	0.0	4.00	5.50	5.50	3.00	1.50	1.50
Step	10	0.0	5.00	4.00	1.00	3.00	6.00	2.00
Quartic	10	0.0	6.00	3.50	1.00	5.00	2.00	3.50
Foxhole	2	0.0	2.00	2.00	6.00	2.00	4.50	4.50
Rastrigin	10	0.0	1.00	5.00	6.00	4.00	3.00	2.00
Schewefel	10	0.0	6.00	5.00	4.00	1.00	2.00	3.00
Griewangk	10	0.0	2.00	4.50	4.50	1.00	4.50	4.50
Ackley	10	0.0	5.00	4.00	6.00	1.00	3.00	2.00
Easom	6	1.0	4.00	2.00	1.00	6.00	3.00	5.00
Schewefel's								
Double Sum	10	0.0	3.50	3.50	3.50	3.50	3.50	3.50
Royal Road	8	0.0	5.00	2.00	6.00	3.00	1.00	4.00
Goldberg	30	0.0	1.00	3.50	5.50	3.50	5.50	2.00
Whitley	6	0.0	5.00	6.00	3.00	2.00	4.00	1.00
avg.			3.81	3.88	4.08	2.92	3.35	2.96

Table 1 Ranks based on success rate and average number of fitness evaluations.

provided. 1 indicates the best hyper-heuristic, while 6 denotes the worst one. These rankings show that the self organising framework improves upon previous versions that embed reinforcement learning. The subtractive punishment rate turns out to be the best choice for negative reinforcement considering the average rank of each hyper-heuristic. This study shows that type D hyper-heuristics have potential. In this study, self organisation of the heuristics is obtained by using the scores generated by reinforcement learning. It is possible to design different self organising hyper-heuristics based on different strategies.

Acknowledgements The experiments were performed at Yeditepe University, Computer Engineering Department, Istanbul, Turkey.

References

1. Burke, E., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research* **176**, 177–192 (2007)
2. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: F. Glover, G. Kochenberger (eds.) *Handbook of Metaheuristics*, pp. 457–474. Kluwer (2003)
3. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics* **9**(6), 451–470 (2003)
4. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. In: M.G.C. Resende, J.P. de Sousa (eds.) *Metaheuristics: Computer Decision-Making*, chap. 9, pp. 523–544. Kluwer (2003)
5. Ozcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive survey of hyperheuristics. *Intelligent Data Analysis* **12**(1), 1–21 (2008)
6. Rattadilok, P., Kwan, R.S.K.: Dynamically configured λ -optimal heuristics for bus scheduling. In: *Proc. of the International Conference on the Practice and Theory of Automated Timetabling (PATAT'06)*, pp. 473–477 (2006)
7. Ross, P.: Hyper-heuristics. In: E.K. Burke, G. Kendall (eds.) *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chap. 17, pp. 529–556. Springer (2005)
8. Ross, P., Schulenburg, S., Marin-Blázquez, J.G., Hart, E.: Hyper-heuristics: learning to combine simple heuristics in bin-packing problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'02*. Morgan-Kaufman (2002)

Multilevel search for choosing hyper-heuristics

Ender Ozcan • Edmund K. Burke

1 Introduction

Hyper-heuristics emerge as a powerful search and optimization tools [1,3]. A traditional hyper-heuristic framework consists of two levels. A hyper-heuristic itself operates at a high level and performs a search over a set of low level heuristics for problem solving. The goal is to raise the level of generality and present methodologies that are applicable to a wide range of problems. Initially, a hyper-heuristic is identified as a heuristic to *choose* or *generate* heuristics. This definition is recursive and from another point of view, it characterizes a more general *multilevel hierarchical search* methodology.

The low level heuristics in a hyper-heuristic can be either *constructive* or *perturbative*. Most of the hyper-heuristics that are used to choose perturbative low level heuristics are single point search strategies. [5] identifies two components in such approaches: *heuristic selection* and *move acceptance*. At each iteration, a heuristic is selected using a predetermined strategy to make a move based on the candidate solution at hand and this process is followed by a move acceptance decision regarding to accept or reject the new resulting solution. These types of hyper-heuristics are referred to as *perturbative hyper-heuristics* (Figure 1). Notice that, it is possible to design both perturbative and constructive low level heuristics for a given problem. Then, an additional level on top of a hyper-heuristic might be needed in order to make use of different types of operators in cooperation. We can still refer to such heuristics as hyper-heuristics within the context of multilevel search.

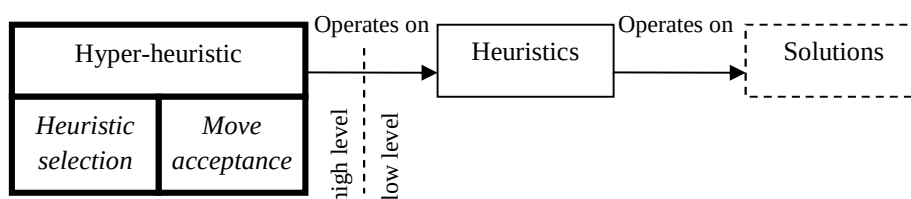


Fig 1. A perturbative hyper-heuristic framework

Ender Ozcan, Edmund K. Burke
University of Nottingham, School of Computer Science
The Automated Scheduling, Optimisation and Planning (ASAP) Research Group
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
E-mails: {exo,ekb}@cs.nott.ac.uk

In [5], the authors define three additional hyper-heuristic frameworks to handle multiple *mutational* and *hill climbing* low level heuristics. Type *D* framework is the most general one that proposes a mechanism to invoke a selected mutational heuristic first and then a selected hill climber. A diversification process takes place in the first phase and then an intensification phase is employed. This framework implicitly performs a multilevel search, where at the top level, the invocation of mutational and hill climbing heuristics are ordered. The use of two different hyper-heuristics is allowed to manage diversifying and intensifying phases. Problem independent information is also allowed to pass in between successive phases. In another study, the author combines different acceptance criteria with a given heuristic selection method under a variety of group decision making frameworks [4]. The proposed group decision making strategies constitute a set of multilevel search techniques. At the top level, a heuristic is employed that tests a set of different hyper-heuristic mechanisms that differ in their acceptance methods to accept or reject a given move.

In this study, we focus on a three-level search methodology that combines multiple perturbative hyper-heuristics as illustrated in Figure 2. We will analyze the performance of a simple multi-level search approach over examination timetabling problems. Examination timetabling problems are hard to solve real world problems [2]. This is an initial study to assess the effectiveness of multilevel search. The initial framework is even further simplified to include the same set of low level heuristics. The results will be reported at the conference.

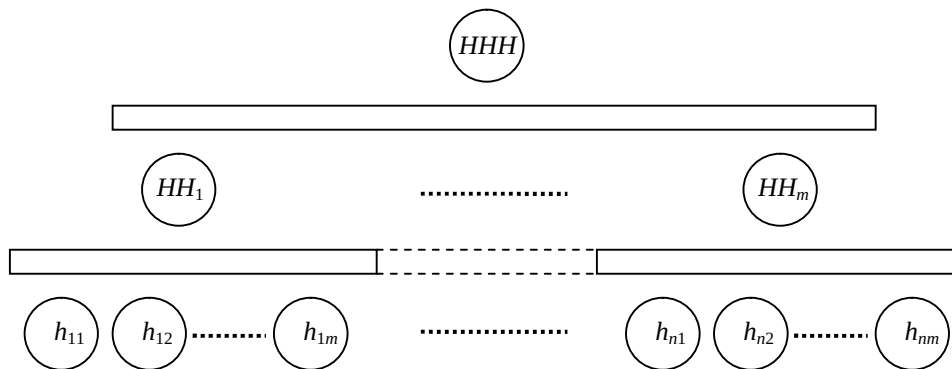


Fig 2. A multilevel search framework with three levels.

References

1. Burke, E. K., Kendall, G., Newall, J., Hart E., Ross, P., Schulenburg, S., Hyperheuristics: An Emerging Direction in Modern Search Technology. In: Glover, F., Kochenberger, G. A. (eds.): Handbook of Metaheuristics. International Series in OR & Management Science, Vol. 57. 457-474, Kluwer Academic Publishers, Boston Dordrecht London (2003)
2. Even, S., Itai, A. and Shamir, A., On the Complexity of Timetable and Multicommodity Flow Problems, SIAM J. Comput., 5(4):691-703 (1976)
3. Özcan, E., Bilgin, B., and Korkmaz, E. E., A comprehensive analysis of hyper-heuristics, Intell. Data Anal., 12:1 , 3-23 (2008)
4. Misir, M, Group Decision Making for Move Acceptance in Hyperheuristics, M.Sc. thesis, Yeditepe University, Department of Computer Engineering (2008)
5. Özcan, E., Bilgin, B., Korkmaz, E. E., Hill Climbers and Mutational Heuristics in Hyperheuristics, Lecture Notes in Computer Science 4193, Springer-Verlag, The 9th International Conference on Parallel Problem Solving From Nature, pp. 202-211 (2006)

HyFlex: A Flexible Framework for the Design and Analysis of Hyper-heuristics

Edmund K. Burke · Tim Curtois · Matthew
Hyde · Graham Kendall · Gabriela
Ochoa · Sanja Petrovic · José Antonio
Vázquez-Rodríguez

1 Introduction

Despite the success of heuristic search methods in solving real-world computational search problems, it is often still difficult to easily apply them to new problems, or even new instances of similar problems. These difficulties arise mainly from the significant number of parameter or algorithm choices involved when using these type of approaches, and the lack of guidance as to how to proceed when selecting them. Hyper-heuristics are an emergent search methodology, the goal of which is to automate the process of either (i) selecting and combining simpler heuristics [5], or (ii) generating new heuristics from components of existing heuristics [6]; in order to solve hard computational search problems. The main motivation behind hyper-heuristics is to raise the level of generality in which search methodologies can operate. They can be distinguished from other heuristic search algorithms, in that they operate on a search space of heuristics (or heuristic components) rather than directly on the search space of solutions to the underlying problem.

The hyper-heuristic framework presented in [5,10], operates at a high level of abstraction and often has no knowledge of the domain. It only has access to a set of low-level heuristics (neighbourhood structures) that it can call upon, and has no knowledge of the functioning of those low-level heuristics. The motivation behind this approach is that once a hyper-heuristic algorithm has been developed, it can be applied to a new problem by replacing the set of low level heuristics and the evaluation function. Figure 1 illustrates that there is a barrier between the low-level heuristics and the hyper-heuristic. Domain knowledge is not allowed to cross this barrier. Therefore, the hyper-heuristic has no knowledge of the domain under which it is operating. It only knows that it has n low-level heuristics on which to call, and it knows it will be passed the results of a given solution once it has been evaluated by the evaluation function. A well defined interface between the hyper-heuristic layer and the problem domain layer needs to be provided, which will allow both the communication between the high-level strategy and the low-level heuristics, and the interchange of relevant non-domain information between the two layers. Furthermore, such an interface would permit the rapid incorporation of new problem domains. In other words, once a new domain is

Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, UK E-mail: {ekb,tec,mvh,gxk,gxo,sxp,jav}@cs.nott.ac.uk

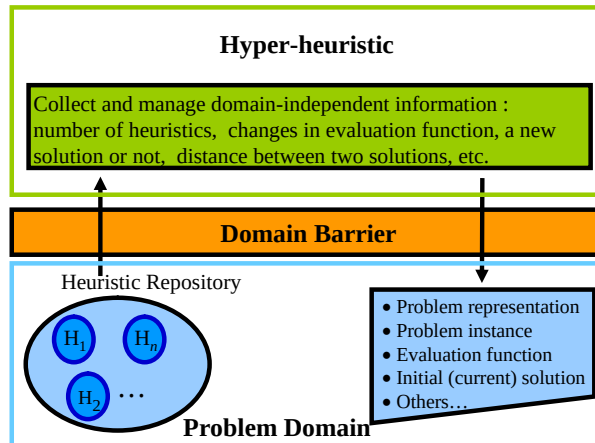


Fig. 1 Hyper-heuristic framework performing single point perturbative search

identified, it would be relatively easy for an expert in the domain to produce a module according to the specifications of the proposed interface.

In this paper we propose a software framework inspired by the hyper-heuristic approach described above. Our goal is to provide software tool for the implementation and comparison of different hyper-heuristics. In doing so, we provide a software interface between the hyper-heuristic and the problem domain layers. The idea of having a software interface for hyper-heuristics was mentioned in [20]. An important feature of our framework, is that we provide the implementation (currently in Java) of a number of diverse combinatorial problem domains, including their solution representation, evaluation function and a set of useful and varied low-level heuristics. These domain modules encapsulate the problem specific algorithm components, and thus liberate the hyper-heuristic designer from knowing the details of the underlying applications. The creative and implementation efforts will instead be focused on the higher-level hyper-heuristic.

Powerful object oriented frameworks for designing and implementing local search heuristics and evolutionary algorithms have been proposed [2, 14, 9]. These frameworks provide a set of modules for implementing the components of search heuristics, leaving the implementation of the problem specific algorithm components to their clients or users. Our HyFlex framework takes the opposite direction! We provide a set of domain modules that encapsulate the problem specific algorithm components, namely, solution representation, evaluation function and a set of low-level heuristics for several hard (real-world) combinatorial problems. What is left to the user is to design high-level strategies (hyper-heuristics) that intelligently combine the set of heuristics provided. We argue that our framework provides a valuable tool for researchers that seek to test their algorithmic ideas on a wide set of problems.

The following section describes in more detail the proposed framework, including the application domains implemented and their associated heuristics.

2 The Proposed Framework

We extended the original perturbative hyper-heuristic framework (Figure 1) in two important ways. First, a memory or list of solutions is maintained in the domain layer, instead of a single incumbent solution. This extension enriches the possibilities for the hyper-heuristic designer, allowing for example the implementation of population based approaches. The idea of providing a list of solutions in a hyper-heuristic framework was first proposed in [25]. Second, a large set of low-level heuristics of different types is provided. Specifically, we consider four types of low-level heuristics, which to the best of our knowledge have not been incorporated simultaneously in a hyper-heuristic framework:

- *Mutational or perturbative heuristics*: perform a (generally) small change in the solution, by swapping, changing, removing, adding or deleting solution components. Note that different mutational heuristics can be easily defined by increasing the extent of the change in the solution.
- *Hill-Climbing heuristics*: iteratively make small changes (mutations or perturbations) to the solution, accepting improving or non-deteriorating solutions, until a local optimum is found or a stopping condition is met. These heuristics differ from mutational heuristics in that they incorporate an iterative improvement process, therefore they guarantee that a non-deteriorating solution will be produced. Different hill-climbing heuristics can be defined by both modifying the extent of the perturbation, or the number of iterations (depth) of the search.
- *Ruin & recreate heuristics*: partly destroy the solution and rebuild or recreate it afterwards. These heuristics can be considered as large neighbourhood structures. They are, however, different from the mutational heuristics in that they can incorporate problem specific constructive heuristics to rebuild the solutions. Different ruin and recreate heuristics can be defined by modifying the extent of the destruction. The construction process can be handled by different constructive heuristics each of which can be used to define a different ruin and recreate heuristic.
- *Crossover heuristics*: combine solution components from two input solutions (parents) to produce a new solution or solutions (offspring). Different variants of a crossover operator can be defined by modifying the proportion of solution components interchanged between parents.

As mentioned above, a number of heuristics of each type can be easily defined by altering the magnitudes controlling their operation. The framework provides two general parameters for defining heuristics: *intensity of change* and *depth of search*. These parameters can be varied in the range $[0, 1]$, and their effects are problem and heuristic dependent. It is the responsibility of the domain module designer to give an adequate meaning to these parameters (together with their default behavior) and properly document his/her choices.

In addition to the low-level heuristics described above, each HyFlex problem domain incorporates:

1. A routine to initialise solutions in the population.
2. A set of interesting instances that can be easily loaded using the method `loadInstance(a)`, where a is the index of the instance to be loaded.

3. A population of one or more solutions that has to be administered.

Currently 4 domain modules are implemented. Namely, the permutation flowshop problem, 1D bin packing, boolean satisfiability and personnel scheduling. Below we overview the main design choices for each domain. Technical reports are available describing the details of each module [11,17,16,24].

2.1 The permutation flow shop problem

The permutation flow shop problem requires finding the order in which n jobs are to be processed in m consecutive machines. The jobs are processed in the order machine 1, machine 2, ..., machine m . Machines can only process one job at a time and jobs can be processed by only one machine at a time. No job can jump over any other job, meaning that the order in which jobs are processed in machine 1 is maintained throughout the system. Moreover, no machine is allowed to remain idle when a job is ready for processing. All jobs and machines are available at time 0. Each job i requires a processing time on machine j denoted by p_{ij} .

Initialization : Solutions are initialized using the well established NEH procedure [19].

This heuristic has been used as an important component of many effective meta-heuristics for the permutation flow shop problem. It has been used as both the initialization procedure of solutions, to be later improved, and also as the improving mechanism within the main iteration of more elaborate algorithms.

Low-level heuristics : A total of 14 low level heuristics were implemented. Specifically, 5 mutational, 4 local search (inspired by those proposed in [21]), 3 crossover heuristics (classical recombination operators for permutation representation) and 2 ruin and recreate heuristics (which incorporate the successful NEH procedure in the construction process). For more details see [24].

Instance data : A total of 120 instances from the widely known Taillard set [23], are provided. The instance sizes are given in Table 1, in the format $n \times m$. The job processing times, on all instances, are uniformly distributed random integers in the range [1,99].

2.2 One dimensional bin packing

The one-dimensional bin-packing problem involves a set of integer-size pieces L , which must be packed into bins of a certain capacity C , using the minimum number of bins possible. In other words, the set of integers must be divided into the smallest number of subsets so that the sum of the sizes of the pieces in a subset does not exceed C .

Initialization : Solutions are initialized by first randomizing the order of the pieces, and then applying the widely known 'first-fit' heuristic [18]. This is a constructive heuristic, which packs the pieces one at a time, each into the first bin that they will fit into.

Low-level heuristics : 2 mutational, 2 ruin and recreate, repacked with best-fit, and 3 local search heuristics. These heuristics are inspired by those proposed in [1]. For more details see [17]

Instance data : The problem instances are summarized in table 2. There are 60 instances in total, 20 in each of three classes.

Instance number	Size
0-9	20×5
10-19	20×10
20-29	20×20
30-39	50×5
40-49	50×10
50-59	50×20
60-69	100×5
70-79	100×10
80-89	100×20
90-99	200×10
100-109	200×20
110-119	500×20

Table 1 Permutation flowshop module instances.

Set name	Piece size distribution	Bin capacity	Number of pieces	Ref
bp1	Uniform [20,100]	150	1000	[12]
bp2	Triples [25,50]	100	501	[12]
bp3	Uniform [150,200]	1000	100	[22]

Table 2 One dimensional bin-packing instance sets. Each set contains 20 instances.

2.3 Boolean satisfiability

The boolean satisfiability or SAT problem involves determining if there is an assignment of the boolean variables of a formula, which results in the whole formula evaluating to true. If there is such an assignment then the formula is said to be satisfiable, and if not then it is unsatisfiable. The process of finding an assignment that satisfies the formula is the search problem considered in this domain module.

Initialization : Solutions are initialized by simply randomly assigning a true or false value to each variable. The problem instances included are examples of the so called 3SAT problem, where each clause contains three variables.

Low-level heuristics : 2 mutational, 4 local search, and 2 heuristics that combine mutation and local search. These heuristics are described in [13], and comprise state of the art local search heuristics for this problem. For more details see [16]

Instance data : The problem instances are taken from the “Uniform Random-3-SAT” category on the ‘SATLIB’ website [15]. There are 60 instances in total, 20 from each of three classes. The instances are summarized in table 3.

2.4 Personnel scheduling

The personnel scheduling problem involves deciding at which times and on which days (i.e. which shifts) each employee should work over a specific planning period. However, the personnel scheduling problem is actually a title for a group of very similar problems.

Instance set name	Variables	Clauses
uf200-860	200	860
uf225-960	225	960
uf250-1065	250	1065

Table 3 Boolean satisfiability module instances.

There is no general personnel scheduling problem. Instead there is a group of problems with a common structure but which differ in their constraints and objectives. This creates an additional challenge in implementing a problem domain module for personnel scheduling. To overcome this we have designed a data file format for which each instance can select a combination of a objectives and constraints from a wide choice. We then implemented a software framework containing all the functions for these constraints and objectives.

Initialization : Initial solutions are created with a hill climbing heuristic which uses a neighbourhood operator that adds new shifts to the roster.

Low-level heuristics : 3 mutational (including *vertical*, *horizontal* and *new* swaps, see [11]), 5 local search, 3 ruin and recreate, and 3 crossover heuristics. These heuristics are taken from previously proposed successful meta-heuristic approaches to nurse rostering problems [3, 4, 7, 8]

Instance data : The instances have been collected from a number of sources. Some of the instances are from industrial collaborators. These include: ORTEC an international consultancy and software company who specialise in workforce planning solutions and SINTEF, the largest independent research organisation in Scandinavia. Other instances have been provided by other researchers or taken from various publications. The collection is a very diverse data set drawn from eleven different countries. The majority of the instances are real world scenarios. An overview of the instances can be found in [11], they vary in the length of the planning horizon, the number of employees and the number of shift types. Each instance also varies in the number and priority of objectives present¹.

3 Discussion

We are proposing a novel framework for supporting research into modern search methodologies. The emphasis of our HyFlex framework lies in providing the algorithm components that are problem specific, thus liberating our users (algorithm designers) from needing to know the problem domain's specific details. The design efforts will instead be focused on designing high-level strategies to intelligently combine the provided tools. Preliminary tests on the reusability of the modules have successfully been conducted. Our motivation is to promote research towards the design of general search methodologies. We plan to extend the number of problem domains and propose a challenge, based on our framework, where the winners will be those algorithms with a better overall performance across all of the different domains. The competition will be 'fair' in that

¹ The instances can be downloaded from:
<http://www.cs.nott.ac.uk/~tec/NRP/>

it will be conducted in a high-level of abstraction: the application domain implementation details are hidden to the framework users. Using an Olympic metaphor, we are no longer interested in the 100 meters race, but instead in conducting the *Decathlon* of modern search methodologies.

References

1. R. Bai, J. Blazewicz, E. K. Burke, G. Kendall, and B. McCollum. A simulated annealing hyper-heuristic methodology for flexible decision support. Technical report, School of Computer Science, University of Nottingham, 2007.
2. S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA—A Platform and Programming Language Independent Interface for Search Algorithms. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632 of *LNCS*, pages 494–508, Berlin, 2003. Springer.
3. E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A scatter search for the nurse rostering problem. Technical report, School of Computer Science, University of Nottingham, 2007.
4. E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A time predefined variable depth search for nurse rostering. Technical report, School of Computer Science, University of Nottingham, 2007.
5. E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 457–474. Kluwer, 2003.
6. E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In C. Mumford and L. Jain, editors, *Collaborative Computational Intelligence*. Springer, 2009. (to appear).
7. E.K. Burke, P. Cowling, P. De Causmaecker, , and G. Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied Intelligence*, 15(3):199–214, 2001.
8. E.K. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2):330–341, 2008.
9. S. Cahon, N. Melab, and E-G. Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, May 2004.
10. P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach for scheduling a sales summit. In *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000*, LNCS, pages 176–190, Konstanz, Germany, 2000. Springer.
11. T. Curtois. A hyflex module for the personnel scheduling problem. Technical report, School of Computer Science, University of Nottingham, 2009.
12. E Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.
13. A. S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation (MIT Press)*, 16(1):31–1, 2008.
14. L. Di Gaspero and A. Schaerf. Easylocal++: an object-oriented framework for the flexible design of local-search algorithms. *Softw. Pract. Exper*, 33(8):733–765, 2003.
15. H. H. Hoos and T. Stützle. Satlib: An online resource for research on sat. In I. P. Gent, H. V. Maaren, and T. Walsh, editors, *SAT 2000*, pages 283–292. IOS Press, 2000. SATLIB is available online at www.satlib.org.
16. M. Hyde. A hyflex module for the boolean satisfiability problem. Technical report, School of Computer Science, University of Nottingham, 2009.
17. M. Hyde. A hyflex module for the one dimensional bin-packing problem. Technical report, School of Computer Science, University of Nottingham, 2009.
18. D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. Worst-case performance bounds for simple one-dimensional packaging algorithms. *SIAM Journal on Computing*, 3(4):299–325, December 1974.
19. M. Nawaz, E. Ensore Jr., and I. Ham. A heuristic algorithm for the m -machine, n -job flow-shop sequencing problem. *OMEGA-International Journal of Management Science*, 11(1):91–95, 1983.

20. A. J. Parkes. A proposal for a hyper-heuristics software interface. Oral presentation, May 2007. Automated Scheduling, Optimisation and Planning Research Group. Internal Seminar.
21. R. Ruiz and T. G. Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *Journal of Operational Research*, 187(10):1143–1159, 2007.
22. P. Schwerin and G. Wäscher. The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp. *International Transactions in Operational Research*, 4(5):377–389, 1997.
23. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.
24. J. A. Vazquez-Rodriguez. A hyflex module for the permutation flow shop problem. Technical report, School of Computer Science, University of Nottingham, 2009.
25. J. Woodward, A. Parkes, and G. Ochoa. A mathematical framework for hyper-heuristics. Oral presentation, 2008 September. Workshop on Hyper-heuristics - Automating the Heuristic Design Process, in conjunction with the 10th International Conference on Parallel Problem Solving From Nature (PPSN X), Dortmund, Germany.

Multilevel Search for Evolving the Acceptance Criteria of a Hyper-Heuristic

Matthew Hyde · Ender Özcan · Edmund K. Burke

1 Introduction

A hyper-heuristic can be defined as a heuristic which searches a space of heuristics, as opposed to directly searching a solution space [1]. A traditional hyper-heuristic framework therefore consists of two levels. The hyper-heuristic itself operates on the higher level, choosing between a set of problem specific heuristics on the lower level, either with a fixed strategy or with a learning mechanism [2,3].

Perturbative hyper-heuristics use a set of perturbative low level heuristics. They can be defined both by their heuristic selection mechanism, and by their move acceptance criteria. The heuristic selection mechanism iteratively chooses which heuristic to apply, given the current problem state. The acceptance criteria decides whether to accept a move generated by this heuristic. For example, [4] presents a tabu search selection mechanism, with an ‘accept all’ acceptance criteria. This selection mechanism could easily be combined with an ‘accept only improving’ acceptance criteria, or a stochastic criteria such as simulated annealing.

Given a fixed heuristic selection mechanism, different acceptance criteria work well on different problems. We show that genetic programming [5] can evolve an effective move acceptance criteria for two problem domains, bin packing [6] and SAT [7,8]. Metaheuristics are popular search techniques for timetabling and scheduling problems, and this technique could be used to automatically design a metaheuristic for a given problem domain.

Genetic programming has been shown to be effective as a hyper-heuristic to generate heuristics over a range of different problem domains, including production scheduling [8–10]. In this case, we will use a similar system to generate the acceptance criteria component of a simple random hyper-heuristic. The genetic programming is therefore operating at a higher level than a hyper-heuristic in a traditional two level framework. The genetic programming is searching a space of hyper-heuristics, not a space of solutions or even problem specific heuristics. It has the characteristics of a more general *multilevel* search methodology, which takes advantage of the recursive nature of the hyper-heuristic definition, ‘heuristics which search a space of heuristics’. We present the

Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, UK. E-mail: [mvh,exo,ekb]@cs.nott.ac.uk

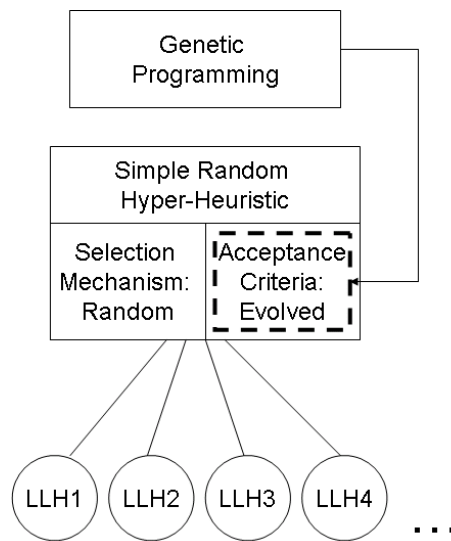


Fig. 1 A multilevel search framework with genetic programming

preliminary results of applying genetic programming as the highest level heuristic in a multilevel search framework, applied to the bin packing and SAT problem domains.

2 Methodology

The simple random heuristic selection mechanism is kept constant, and we compare the performance of both an ‘accept all’ and a great deluge acceptance criteria, with an acceptance criteria evolved with genetic programming. Our aim is to show that genetic programming is capable of evolving an acceptance criteria which is specialised to a given problem. These experiments are motivated by the observation that ‘accept only improving’ is more a effective hyper-heuristic acceptance criteria for bin packing than for SAT, while the converse is true for an ‘accept all’ strategy.

Figure 1 shows the overall structure of the framework. The genetic programming evolves a population of acceptance criteria to be used in the simple random hyper-heuristic. To assess the fitness of an acceptance criteria within the genetic programming algorithm, the simple random hyper-heuristic is run on a problem instance with the acceptance criteria. The hyper-heuristic repeatedly chooses a low-level heuristic (LLH’s in figure 1) to apply, which is applied to the current solution. The new solution is accepted if the GP individual returns a value less than 0.5 when evaluated. It is important to note that even improving solutions are accepted based on the result of the GP individual. The hyper-heuristic is asked to solve five instances, and the sum of their objective function values is assigned to the individual as its fitness value.

2.1 Implementation Details

The low-level heuristics that we have implemented for these two problems are well known problem specific heuristics from the literature. Independent sets of low-level

Terminals	Functions
Delta (change in fitness)	+
Current Fitness	−
Previous Fitness	*
Current Time	% (Protected Divide)
Total Time	e^x

Table 1 The functions and terminals included in the genetic programming system

heuristics are used for bin packing and SAT, and there are seven implemented for bin packing and eight implemented for SAT. The heuristics are all perturbative heuristics, they start with a feasible solution and return a new feasible solution.

The 100 problem instances included for bin packing contain 100 pieces, uniformly distributed between 150 and 200. The bin capacity is set to 1000. These instances can be found at <http://paginas.fe.up.pt/~esicup/tiki-index.php>. The 100 problem instances included for SAT are the ‘uf250-1065’ instances available at <http://www.satlib.org/>, and they are all satisfiable.

2.2 Genetic Programming Parameters

The acceptance criteria are represented as tree structures, as in canonical genetic programming. The population size is set to 500, and each run lasts for 40 generations. The possible components of the trees (the functions and terminals) are shown in table 1. Protected divide (%) changes a division by zero into a division by 0.001. The fifth function is Euler’s number raised to the power of the result of the child node, and as such this is the only function with just one child node. Delta is the change in fitness from the last time that a solution was accepted. The current time is the elapsed time, in milliseconds, since the individual began solving the instance. The total time is effectively a constant of 5000, as each hyper-heuristic individual is allowed to run for five seconds.

3 Preliminary Results

A simple random strategy is employed as the heuristic choice mechanism in all experiments. We use 100 instances from the literature for each problem, and we use five to train a heuristic. 95 instances are therefore used for testing, and the average result of these is reported.

Table 2 shows a summary of the time taken to evolve one heuristic. The fitness of a heuristic is the sum of its results on the five training instances. Each heuristic is run for five seconds, on each of five instances, so it takes 25 seconds to give a fitness score to a heuristic in the population. We evolved 30 heuristics in parallel for each of the two problem domains.

Table 3 presents the results. The figures shown represent the average result over the 95 instances in the test set. For bin packing, the evaluation function is shown in equation 1, where: n = number of bins, $fullness_i$ = sum of all the pieces in bin i , and C = bin capacity. This function returns a value between zero and one, where lower is

Population	500
Generations	40
Instances	5
Time per Instance	5 secs
Total Seconds	500000
Total Hours	138.9

Table 2 The time taken to run the experiments

	Bin Packing	SAT
SR-AM	0.133	5.158
SR-GD	0.078	5.147
Best Evolved	0.063	4.316
Evolved Average	0.065	5.595

Table 3 Comparison of evolved versus fixed acceptance criteria, a lower result is better

better, and a set of completely full bins would return a value of zero.

$$Fitness = 1 - \left(\frac{\sum_{i=1}^n (fullness_i/C)^2}{n} \right) \quad (1)$$

The evaluation function for the SAT problem is simply the number of broken clauses. Therefore, a lower value is better, and a value of zero represents a satisfied formula.

Results are shown for the best heuristic and for the average of the 30 heuristics that were evolved. They show that for both the bin packing and SAT problem domains, genetic programming is able to evolve an acceptance criteria that works better than either a great deluge or ‘accept all’ strategy.

References

1. Ross, P.: Hyper-heuristics. In: E.K. Burke, G. Kendall (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, pp. 529–556. Kluwer, Boston (2005)
2. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis* **12**(1), 3–23 (2008)
3. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: F. Glover, G. Kochenberger (eds.) *Handbook of Meta-Heuristics*, pp. 457–474. Kluwer, Boston, Massachusetts (2003)
4. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics* **9**(6), 451–470 (2003)
5. Koza, J.R.: *Genetic programming: on the programming of computers by means of natural selection*. The MIT Press, Boston, Massachusetts (1992)
6. Martello, S., Toth, P.: *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, Chichester (1990)
7. Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI’92)*, pp. 440–446. San Jose, CA, USA (1992)
8. Fukunaga, A.S.: Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation (MIT Press)* **16**(1), 31–1 (2008)
9. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In: *Proceedings of the 9th ACM Genetic and Evolutionary Computation Conference (GECCO 2007)*, pp. 1559–1565. London, UK. (2007)
10. Geiger, C.D., Uzsoy, R., Aytug, H.: Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling* **9**(1), 7–34 (2006)

Scheduling of job shop, make-to-order industries with recirculation and assembly: discrete versus continuous time models

Marta C. Gomes, Ana Barbosa-Póvoa and Augusto Q. Novais

Abstract This work studies the performance of two Mixed Integer Linear Programming (MILP) models to solve scheduling problems in a flexible job shop environment with recirculation and assembly using a due-date-based objective function. The models convey different approaches both in the modelling of time (discrete and continuous approaches) as well as in the assignment of jobs to machines. The comparison is carried out for a job shop system considered closer to the industrial reality than the classical job shop problem of a single machine per operation that has been extensively studied in the literature, with the mould making industry providing the motivating application.

1 Introduction

A commonly used classification divides mathematical programming models for scheduling into discrete and continuous time models. Discrete time models divide the scheduling horizon into a finite number of intervals with equal and predefined duration and allow the beginning and ending of operations to take place only at the boundaries of these time periods. In continuous time models there is no previous division of the scheduling horizon, and timing decisions are explicitly represented as a set of continuous variables defining the exact times at which the events take place. Models of both kinds have been intensively developed for scheduling problems in the process industries but their application in discrete parts manufacturing has been quite limited so far.

A mould is a specially designed tool consisting of a “base” and one or more “cavities” contoured to the exact specifications of the desired product. A moulding compound is filled into the cavities, hardens by cooling and/or by completion of a chemical reaction and is extracted once it has taken the shape of the mould. A closer look at our daily environments shows the ubiquity of moulded pieces: the automotive, household and electrical appliances industries, together with the packing, electronics, telecommunications and hardware industries are the main clients of the mould industry. Mould makers operate in a make-to-order basis, producing one-of-a-kind products (orders seldom consist of more than one copy of a mould). Machinery is organized in the shop floor as in a typical job shop. The Mixed Integer Linear Programming formulations developed model explicitly several features of the mould fabrication process:

Marta C. Gomes
CESUR (Centre for Urban and Regional Systems), Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisboa, Portugal
E-mail: marta.gomes@ist.utl.pt

Ana Barbosa-Póvoa
CEG-IST (Centre for Management Studies), Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisboa, Portugal
E-mail: apovoa@ist.utl.pt

Augusto Q. Novais
Department of Process Modelling and Simulation, Instituto Nacional de Engenharia,
Tecnologia e Inovação, Est. Paço do Lumiar, 1649-038 Lisboa, Portugal
E-mail: augusto.novais@ineti.pt

- machine groups composed of multiple machines operating in parallel (flexible job shop);
- generalized processing routes, with no constraints upon the machine groups that constitute them;
- recirculation: some jobs visit a machine group more than once;
- assembly of components produced separately to form an order;
- every order has a due date for delivery to the customer.

In the *continuous time formulation*, the decision variables in common with the model of Manne (1960) are the starting time of each job on each machine (continuous) and the sequencing variables (binary) that establish the precedence between tasks in each processing unit. The notion of global precedence is used instead of immediate precedence. The model additionally accounts for product recirculation, product combination to form an order and a due date-based objective function. Orders may correspond to a single component or to several components combined in a final assembly operation. Earliness and tardiness variables were added following the approach of Zhu and Heady (2000) for the earliness and tardiness problem in a single-stage system of non-identical parallel units. While Manne's model considers a single machine per operation, non-identical machines per operation and binary assignment variables were considered following the approach of Méndez *et al.* (2001) in solving a flowshop problem in the process industry.

Scheduling objectives are: to finish each order as close as possible to the corresponding due date and to minimize storage of unfinished components in intermediate buffers (the queues to the machines). Earliness and tardiness costs for the orders are given, as well as costs of intermediate storage of unfinished components.

The *discrete time scheduling formulation* developed is based on the one presented by Chang and Liao (1994) for flexible flow-shop scheduling, which comprises homogeneous machine groups, limited buffers and orders composed of several identical units that follow the same processing route. A number of units (of the same or different orders) may be processed at the same time in a machine group. Gomes *et al.* (2005) extended this model to the flexible job shop case with generalized processing routes and recirculation. The model was further generalized to consider the existence of non-homogeneous machine groups (machines in a group may have different processing times). Orders corresponding to single or assembled components are considered (the notion of "unit" is replaced by "component" in the new model) and assembly operations were added (Gomes 2007). The scheduling horizon is divided into identical time slots; all model variables are integer.

In both models, the objective function (to be minimized) is a weighted sum of order earliness, tardiness and intermediate storage time. To summarize, in the discrete time formulation, time is modelled explicitly through a time index with no binary variables being required for machine assignment. In the continuous time formulation, modelling of time is implicit and jobs are assigned to machines through binary variables.

2 Computational study

The example considered is based on data collected at a mould making plant. Figure 1 is a simplified diagram of the plant: it consists of fifteen machines (M1 to M15) and four processing routes (A,B,C,D) that the components may follow. Orders are formed either by individual components or by the assemblage of two components, with two possibilities: a component from route A combined with a component from route C (assembly 1, performed in "machine", or assembly unit M14) or a component from route B joined with a component from route D (assembly 2, performed in assembly unit M15). Routes B and C display recirculation: machines M3, M10 and M11 are visited twice.

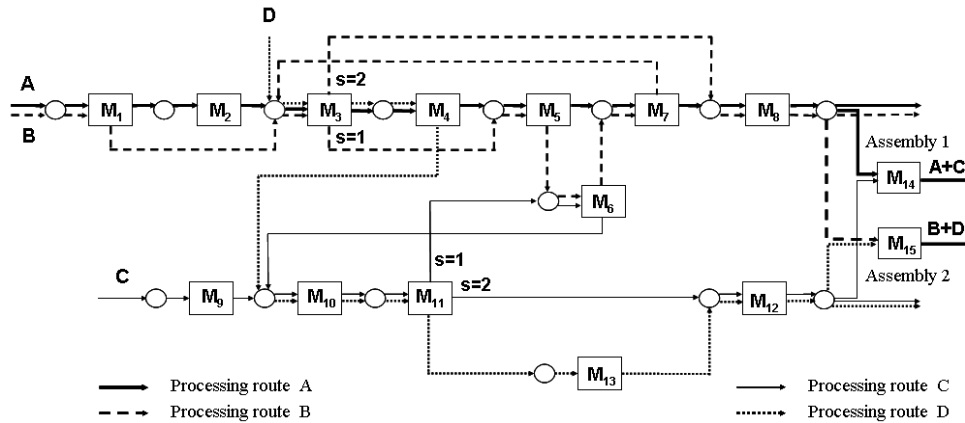


Figure 1. Processing routes in a shop producing single and assembled orders.

Five sets of orders to be scheduled were generated, involving 10, 20, 30 and 40 components (which correspond to “jobs” in the job shop terminology). Each set is composed of single and assembled orders. Two sets of 40 components were generated, which differ in the average interval between consecutive due dates: a “tight” due date set (40A) and a “loose” due date set (40B). The simplest operational scenarios considered have one or two machines per machine group. Scenario 2+(3,4) displays 34 machines in total: two machines per machine group, recirculation groups M10 and M11 with three machines each, and recirculation group M3 with four machines. Likewise for scenario 4+(5,6), with 64 machines in total: four machines per machine group, groups M10 and M11 have five machines each and group M3 has six machines. A complete description of order data and processing times can be found in Gomes (2007).

The MILP models were implemented in GAMS modelling system and solved with CPLEX on a 3 GHz Pentium IV with 512MB RAM running Windows XP Professional. CPLEX releases 9.1.2 (10 component set) and 10.0.1 (other sets) were used. Table I summarizes the results, by displaying model statistics, computational performance, the objective function value and also the total number of operations. “Gap” is the optimality gap (in percentage). For the continuous time model, a better estimate of the gap than the one returned by the solver was determined by using the optimal solution or the lower bound computed by the discrete time model, whenever it was available. For models not solved to optimality, “solution polishing” was performed (a heuristic procedure available in CPLEX 10 that improves a solution after the branch-and-bound search).

2.1 Discussion of results

When analyzing the results (see Table I) for the 10 and 20 component sets, the continuous time model performed better than its discrete counterpart when a single machine is available per operation. With the continuous time model either the optimal solution was computed in a shorter time (10 component set) or a better sub-optimal solution could be obtained with less computational effort (20 component set; 422 sec against 2701 sec).

However, when assignment to machines is considered (scenarios 2 and 2+(3,4) machines), the discrete time formulation generally outperformed the continuous one although the model size (variables plus constraints) is considerably larger. In all problem instances, except set 40A, the discrete time model was solved in CPU time less or equal to 12.7 min (759 sec). The continuous time model either took, on average, longer time to be solved or could not

Table I
Computational statistics and scheduling results

Scenario	No. of components	No. of variables	No. of constraints	Objective Function	Gap (%)	CPU (sec)	No. of operations
<i>Continuous time model</i>							
1 machine	10	329	477	621.8	0	4.3	71
	20	1,067	1,775	755.9	11.8	422.1	141
2 machines	10	400	801	0.8	0	192.6	71
	20	1,208	3,247	0.5	20.0	2700.2	141
2+(3,4) machines	20	1,278	4,387	0.2	0	40.7	141
	30	2,510	9,863	0.3	33.3	1200.3	210
	40A	4,198	17,845	81.8	100	1203.4	282
	40A	“	“	22.3	100	3000.3	282
	40B	“	“	1.2	100	1200.4	282
4+(5,6)	40A	4,762	30,305	0	0	356.4	282
<i>Discrete time model</i>							
1 machine	10	18,279	13,562	621.8	0	27.4	71
	20	52,025	33,958	859.9	22.5	2701.3	141
2 machines	10	27,233	16,356	0.8	0	9.6	71
	20	77,162	38,935	0.4	0	95.9	141
2+(3,4) machines	20	89,132	40,345	0.2	0	31.6	141
	30	132,269	55,962	0.2	0	248.8	210
	40A	178,263	73,156	— ^(a)	—	—	282
	40B	220,743	89,356	0	0	758.6	282

^(a) An integer solution could not be found within the time limit of 30 min for branch & bound.

prove optimality within a time limit of 10 min (or longer) for branch-and-bound and 10 min (or longer) for solution polishing.

In set 40A of “tight” due dates and operational scenario 2+(3,4) no integer solution to the discrete formulation was found within 30 min of running the solver. Nevertheless, the continuous time formulation found a feasible solution in this case: a double entry “40A” is shown which corresponds to two solutions obtained with different time limits for branch-and-bound and solution polishing. A reduction of 73% in the objective function value was achieved when total run time increased from 20 to 50 min (1200 to 3000 sec).

Finally the table shows that two problem instances that could not be solved to optimality in 30 min of branch-and-bound with the continuous time model can be so if the number of machines per group is increased (and despite the increase in model size). This is the case of the 20 component set when the number of machines is increased from 2 to 2+(3,4) (41 sec to obtain the optimal solution) and the 40A component set when the operational scenario varies from 2+(3,4) to 4+(5,6) (356 sec or 5.9 min of CPU time).

As a main conclusion it can be stated that in the operational scenario of a single machine per operation the continuous time formulation performs better than its discrete counterpart while in the scenario of two or more machines in parallel per operation the discrete time formulation outperforms the continuous one. This result contrasts with the neglect of discrete time models in the fields of flow shop and job shop scheduling, where continuous time MILP

formulations are considered superior based on the performance on classical models of one machine per operation (Pan (1997), Stafford *et al.* (2005), Pan and Chen (2005)).

Results also show that model performance is not directly related to the model size, but critically depends on factors like due date distribution and the number of machines available per operation. In spite of the continuous time model being generally outperformed by its discrete counterpart, there was a large size instance where no integer solution of the discrete model was found within 30 min of running the solver. In such situations the continuous time model becomes more attractive because a feasible solution can be obtained and further enhanced in relatively short time.

3 Conclusion and future developments

The development and comparison of discrete and continuous time formulations for solving the job shop scheduling problem is an original contribution of this study. Pan (1997) compares different formulations for the classical job shop problem in terms of model size complexity (number of variables and number of constraints) but no experimental study seems to have been reported like the one of Stafford *et al.* (2005) for the classical permutation flow shop problem.

The models developed apply directly to different types of job shop environments; this is the case of the mould making industry. Portugal has a relevant place in the world's mould producers. Manufacturers employ processes and techniques with a strong technological level and are internationally recognized for the technical quality and the ability to adapt to new and increasingly complex challenges. However, among others weaknesses of the sector, the following have been pointed out: poor interaction with knowledge centres, difficulty in meeting due dates and few highly competitive companies coexisting with a vast number of firms with ineffective organization and out-of-date equipments. In this context, mould makers would benefit from the use of decision support systems (DSS) to organize production. This work expects to be a contribution towards development of such decision tools.

As future developments, on the one hand the conclusions of this research should be consolidated by solving more problem instances and studying other configurations of the job shop system. On the other hand, solving larger and more realistic problems will pose the challenge of obtaining efficient and fast solutions for MILP models. Several techniques have been proposed with the aim of maintaining the number of decisions at a reasonable level in exact approaches of large-scale problems (Méndez *et al.* 2006). In the near future we intend to explore model reduction by means of heuristics. The incorporation of structural knowledge of the problem into the mathematical representation (like simple or combined dispatching rules) can lead to reduced models that capture only the critical decisions to be made and therefore to good solutions that can be generated in reasonably short time.

References

1. Chang, S.-C. and Liao, D.-Y., "Scheduling flexible flow shops with no setup effects", *IEEE Transactions on Robotics and Automation*, vol. 10, pp. 112-122 (1994).
2. Gomes, M.C., Barbosa-Póvoa, A. and Novais, A.Q., "Optimal scheduling for flexible job shop operation", *International Journal of Production Research*, vol. 43, pp. 2323-2353 (2005).
3. Gomes, M.C., *Reactive scheduling in make-to-order production systems: An optimization based approach*, Ph.D. dissertation in Systems Engineering, Instituto Superior Técnico, Universidade Técnica de Lisboa (2007).
4. Manne, A. S., "On the job-shop scheduling problem", *Operations Research*, vol. 8, pp. 219-223 (1960).

5. Méndez, C.A., Henning, G.P. and Cerdá, J., "An MILP continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities", *Computers and Chemical Engineering*, vol. 25, pp. 701-711 (2001).
6. Méndez, C.A., Cerdá, J., Grossmann, I.E., Harjunkski, I. and Fahl, M., "State-of-the-art review of optimization methods for short-term scheduling of batch processes", *Computers and Chemical Engineering*, vol. 30, pp. 913-946 (2006).
7. Pan, J.C-H., "A study of integer programming formulations for scheduling problems", *International Journal of Systems Science*, vol. 28, pp. 33-41 (1997).
8. Pan, J.C-H. and Chen, J-S., "Mixed binary integer programming formulations for the reentrant job shop scheduling problem", *Computers and Operations Research*, vol. 32, pp. 1197-1212 (2005).
9. Stafford Jr., E.F., Tseng, F.T. and Gupta, J.N.D., "Comparative evaluation of MILP flowshop models", *Journal of the Operational Research Society*, vol. 56, pp. 88-101, (2005).
10. Zhu, Z. and Heady, R.B., "Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach", *Computers and Industrial Engineering*, vol. 38, pp. 297-305 (2000).

New results for the multi-objective sequence dependent setup times flowshop problem

Michele Ciavotta · Rubén Ruiz · Gerardo Minella

Although many papers deal with the permutation flowshop scheduling problem with or without setups, according to our knowledge, little has been published tackling multi-objective optimization in presence of sequence dependent setup times. Hence, in this work we cope with this problem considering two pairs of well known independent objectives, the $C_{max} - TWT$ (*Makespan-Total Weighted Tardiness*) and $C_{max} - TFT$ (*Makespan-Total Flowtime*). An effective algorithm, RIPG (*Restarted Iterated Pareto Greedy*), has been developed to face this complex scheduling setting. The RIPG is a Pareto evolution of the IG (*Iterated Greedy*) algorithm, a rather new metaheuristic approach which has shown state-of-the-art performance in single objective optimization for the permutation flowshop problem with [1] and without setups [2]. In essence, it consists of a greedy strategy iteratively applied over an archive of nondominated solutions. The greedy procedure used is an evolution of the well known NEH heuristic [3] and makes use of the Pareto relationship to generate a whole set of nondominated solutions. The rationale of the proposed method is very simple. Roughly, it is possible to divide it into five phases. The first phase is the *Initialization*, where an initial set of good solutions is generated using a heuristic approach. The remaining four phases are iteratively repeated and constitute the bulk of the algorithm. They are: the *Selection* phase, where one or more solutions, belonging to the current archive, are selected for the following steps. A modified version of the *Crowding Distance Assignment* procedure, originally presented in [4], has been developed in order to carry out the selection process. The *Pareto greedy improvement* phase is then applied over the selected solution and it returns a set of solutions which do not dominate each other. During this step, the current solution is disrupted (*Destruction*), removing some jobs from the sequence, and a greedy procedure (*Construction*) is applied. The construction procedure reinserts the eliminated jobs into partial sequences similarly to the insertion procedure of the NEH heuristic, returning, a hopefully improved, nondominated solution set. This set is then added to the current Pareto archive and dominated elements are discarded. A *Local search* phase is hence applied on a selected solution to enrich the process and to improve the current Pareto set in terms of spread and diversity. Lastly, a *Restart*

Michele Ciavotta, Rubén Ruiz, Gerardo Minella
Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Valencia,
E-mail: {mciavotta,mgerar}@iti.upv.es, rruiz@eio.upv.es

procedure is implemented to prevent the algorithm from getting stuck in local optima. In this work we compare our algorithm against the highest performing approaches (especially those proposed for the multi-objective flowshop problem). In [5] a large number of papers dealing with multi-objective flowshop algorithms have been reviewed and the best performing ones have been also implemented. Starting from this point, we first reduced the number of methods by means of a preliminary test on a small set of instances and, depending on the results obtained, we selected the ten best algorithms. Notice that such methods also show the best performance for multi-objective permutation flowshop without setups. During this preliminary phase we noted that the algorithm proposed by [6] (MOSA_Varad) achieved worse results with respect to the others, because it is not able to fully exploit the available CPU time. Therefore, we decided to implement an improved version of this method (which we refer to as MOSA_Varad_M). In [7] a promising algorithm (MOIGS) has been recently proposed and we decided to implement and evaluate.

In all the experiments that we present in this work, we make use of two different instance sets (SDST50 and SDST125), presented in [2]. Each set contains 110 instances with several combinations of the number of jobs n and number of machines m . The $n \times m$ combinations are: $\{20, 50, 100\} \times \{5, 10, 20\}$, $200 \times \{10, 20\}$. Setup times are selected to be respectively 50% and 125% of the processing times (p_{ij}). As regards the performance measures, the comparison of two different Pareto approximations is not straightforward. However, recent studies point out how the so-called “Pareto-compliant” measures (see [8,9]) seem to be the most appropriate. Among these, we selected the hypervolume (I_H) and the multiplicative unary epsilon (I_ϵ^1) indicators which represent the state-of-the-art as far as quality indicators are concerned (for more details on the application of these measures to multi-objective flowshop problem see [5]).

The stopping criterion for all algorithms is given by a time limit that is not fixed but depends on the size of the considered instance. The algorithms are stopped after a CPU running time of $n \cdot m/2 \cdot t$ milliseconds, where t is an input parameter. In this way we can assign more time to larger instances that are, obviously, more time consuming. Every algorithm is run 10 different independent times (replicates) on each instance with two different stopping criteria: $t = 150$ and $t = 200$ milliseconds. A total of 114,400 data points are collected (13 tested algorithms $\times$ 220 instances $\times$ 10 replicates per instance $\times$ 2 different stopping time criteria $\times$ 2 pairs of objectives). Actually, each data point is an approximated Pareto front containing a set of vectors with the two objective values.

Table 1 contains average results for (SDST125) instance set and $(C_{max} - TWT)$ as objectives. Although each depicted value is attained by means of a very large number of samples, it is still necessary to carry out a comprehensive statistical experiment to assess if the observed differences in the average values are statistically significant. A total of 32 different experiments are carried out. We did parametric ANOVA analyses as well as non-parametric Friedman rank-based tests on both quality indicators and for the two different stopping criteria. The utility of using both parametric and non-parametric tests consists in improving the soundness of our conclusions. We carried out 16 (8 for each couple of objectives) multi-factor ANOVAs where the type of instance is a controlled factor. The algorithm is another controlled factor with 13 levels. The response variable on each experiment is either the hypervolume or the epsilon indicator. Lastly, there is one set of experiments for each stopping time. All the tests are carried out with confidence level $\alpha = 0.05$. Considering that each experiment contains 14,300 data points, the three main hypotheses of ANOVA: normality, homoscedasticity

and independence of the residuals are easily satisfied. To compare results, a second set of 16 experiments are performed. In this case, non-parametric Friedman rank-based tests are carried out. Since there are 13 algorithms and 10 different replicates, the results for each instance are ranked between 1 and 130. All these tests proved that IPG widely outperforms all other algorithms for both hypervolume (I_H) and epsilon indicators (I_ϵ^1). As a consequence, IPG can be considered the state-of-art for this important scheduling problem.

SDST125	Time	150		200	
#	Method	I_H	I_ϵ^1	Method	I_H I_ϵ^1
1	RIPG	1.307	1.077	RIPG	1.322 1.067
2	MOIGS	1.194	1.171	MOIGS	1.207 1.160
3	MOGALS_Arroyo	0.980	1.271	MOGALS_Arroyo	0.993 1.260
4	MOSA_Varad_M	0.939	1.348	MOSA_Varad_M	0.949 1.344
5	MOTS	0.930	1.318	MOTS	0.938 1.312
6	PESAI	0.923	1.267	PESAI	0.936 1.262
7	PESA	0.922	1.264	PESA	0.933 1.259
8	MOSA_Varad	0.853	1.404	MOSA_Varad	0.847 1.408
9	PGA_ALS	0.840	1.354	PGA_ALS	0.842 1.352
10	PILS	0.776	1.410	PILS	0.81 1.385
11	MOGA_Murata	0.737	1.389	MOGA_Murata	0.747 1.384
12	ϵ -NSGAII	0.679	1.419	ϵ -NSGAII	0.688 1.416
13	CMOGA	0.669	1.422	CMOGA	0.686 1.413

Table 1 Results for $C_{max} - TWT$ criteria and SDST125 instance set.

References

1. Ruiz, R., Stützle, T.: An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research* **87**(3), 1143–1159 (2008)
2. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research* **177**(3), 2033–2049 (2007)
3. Nawaz, M., Enscore, J.E.E., Ham, I.: A heuristic algorithm for the m machine, n job flowshop sequencing problem. *Omega-International Journal of Management Science* **11**(1), 91–95 (1983)
4. Deb, K.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* **6**(2), 182–197 (2002)
5. Minella, G., Ruiz, R., Ciavotta, M.: A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. *INFORMS Journal on Computing* **20**(3), 451–471 (2008)
6. Varadharajan, T., Rajendran, C.: A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs. *European Journal of Operational Research* **167**(3), 772–795 (2005)
7. Framinan, J., Leisten, R.: A multi-objective iterated greedy search for flowshop scheduling with makespan and flowtime criteria. *OR Spectrum* **30**, 787–804(18) (2008)
8. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation* **3**(4), 257–271 (1999)
9. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G.: Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation* **7**(2), 117–132 (2003)

Exact methods for the resource levelling problem

Thorsten Gather · Jürgen Zimmermann

1 Introduction

Project management in the present time is characterized by rapid advancements in many technologies, hardly calculable risks and very tight deadlines for nearly all kinds of projects. Planning and steering of projects under these conditions turns out to be a really tough job even for the most experienced project managers. To deal responsibly with project sponsors or customers, they must consider carefully the resources that are required to complete a project on time and within the estimated budget. As a consequence, the levelling of resources has faded into the focus of attention. Especially within the mid-term planning a deliberate resource allocation plan is important to avoid bottlenecks in the subsequent short-term resource planning or even during the execution of some project. Resources in this context could for example be employees, machinery or even scarce work spaces (like test stands in automotive development projects).

An interesting application for resource levelling has been proposed in [3]. In make-to-order production all products are manufactured on the basis of a given set of customer orders with individual delivery dates. The production quantities of the final products are therefore externally given due to these orders. A so called Master Production Schedule (MPS) specifies the exact amounts of all main products and the corresponding times of production. The MPS is thereby usually determined in the mid-term planning and serves as input for the subsequent short-term planning.

It has already been discussed in the literature that levelling resource requirements in the mid-term planning of projects is crucial to achieve good feasible solutions at subsequent planning steps because remarkable fluctuations in the utilization of resources like manpower or machinery often cause extra labor or financial expenditure (cf. [5]). In make-to-order production products are only produced on demand and hence the workload distribution over time cannot be balanced via inventory holding. Instead,

Thorsten Gather
Clausthal University of Technology, Germany
E-mail: thorsten.gather@tu-clausthal.de

Jürgen Zimmermann
Clausthal University of Technology, Germany
E-mail: juergen.zimmermann@tu-clausthal.de

resource levelling means to schedule the production orders in a way that a smooth resource utilization is achieved. The particular importance of resource levelling arises out of the non-repetitive character of production orders, that makes it impossible to avoid bottlenecks only by configuring the capacities of the production system in short-term planning.

To model the make-to-order production planning problem [3] proposed a representation of each production order as one activity of a customer project. In this paper we will model general projects or customer projects out of some make-to-order production scenario as a *resource levelling project scheduling problem*.

There have only been a few exact approaches introduced in the open literature to level resource profiles. [1] proposed a method that enumerates all combinations of activity start times for networks with precedence constraints to minimize the squared changes in the resource utilization. This idea has been seized by us [6] for networks with generalized temporal constraints and several objective functions. [5] presented a linear programming approach minimizing the deviations from an average resource level and [4] applied non-serial dynamic programming and interaction graph theory to find a minimum for the squared deviation from the average resource utilization. Finally, [12] presented a tree-based enumeration approach for the *resource renting problem*, which is very similar with regard to its structural properties. In [10] we outlined how this approach can be used to solve the resource levelling problem. In the contribution at hand we will now sketch some weaknesses of the latter approach and develop based on a concept of [7] a new and more efficient procedure.

Since the approaches of [5] and [4] also address networks with only precedence constraints they are not suitable for a direct comparison. In our computational study we will therefore compare our new procedure to our start time enumeration approach [6] - which covers the idea of [1] - and to the approach of [12].

2 Problem description

The paper at hand considers a project scheduling problem with generalized temporal constraints that have for instance been described in [8]. The project under consideration is given by an activity-on-node network N with activity set $V := \{0, 1, \dots, n, n+1\}$, arc set $E \subset V \times V$ and arc weights δ_{ij} . The fictitious activities 0 and $n+1$ represent the beginning and completion of the project, respectively. Let $p_i \in \mathbb{Z}_{\geq 0}$ be the duration of activity $i \in V$, which is assumed to be carried out without interruption. Moreover, let $S_i \geq 0$ be the start time of activity $i \in V$. Given $S_0 := 0$ (i.e., the project always begins at point in time zero), S_{n+1} represents the duration of the project. A vector $S = (S_0, S_1, \dots, S_{n+1})$ with $S_i \geq 0$ ($i \in V$) is called a *schedule*. Temporal constraints between pairs of activities are specified by minimum and maximum time lags. If there is a minimum time lag d_{ij}^{min} ($i, j \in V$) to be observed between two activities i and j , network N contains an arc $\langle i, j \rangle$ with weight $\delta_{ij} := d_{ij}^{min}$. If there is a maximum time lag d_{ij}^{max} ($i, j \in V$), N contains an arc $\langle j, i \rangle$ with weight $\delta_{ji} := -d_{ij}^{max}$. As a special case we term a minimum time lag d_{ij}^{min} with $d_{ij}^{min} = p_i$ *precedence constraint*. A maximum duration $\bar{d}$ for the project is represented by a maximum time lag between the fictitious start and end activities that is, by arc $\langle n+1, 0 \rangle$ with weight $\delta_{n+1,0} := -\bar{d}$.

The set of schedules satisfying all temporal constraints $S_j - S_i \geq \delta_{ij}$ given by minimum and maximum time lags, is denoted by $\mathcal{S}_T$.

Due to the temporal constraints each activity $i \in V$ has an earliest start time ES_i and a latest start time LS_i that can be determined by computing the longest path between node 0 and node i (ES_i) and between i and 0 (LS_i), respectively. The earliest and latest completion times of activity i are given by $EC_i = ES_i + p_i$ and $LC_i = LS_i + p_i$. If some activity has a time window greater than 0 time units, i.e. $LS_i - ES_i > 0$, we speak of a positive *total float* TF_i for activity i .

In order to measure the quality of a certain schedule $S \in \mathcal{S}_T$ we minimize the squared resource utilizations for each resource $k \in R$. Let $c_k \geq 0$ be a cost incurred per utilized unit of resource $k \in R$ and per time unit and furthermore r_{ik} be the amount of resource k that is needed to carry out activity i . Then we can express the total demand of resource k at point in time t for a given schedule S by a so called resource profile $r_k(S, t)$. An appropriate objective function in order to achieve a smooth resource utilization is

$$f(S) := \sum_{k \in R} c_k \int_0^{\bar{d}} r_k^2(S, t) dt \quad (\text{RL})$$

which represents the total squared utilization cost for schedule S (cf. [10]).

Our *project scheduling problem* now consists of minimizing function RL over the set of all time-feasible schedules, i.e.

$$\left. \begin{array}{l} \text{Minimize } f(S) \\ \text{subject to } S \in \mathcal{S}_T \end{array} \right\} \quad (P)$$

As we have shown in [10] problem P is NP-hard even if we consider only one resource. The paper at hand considers a new tree-based enumeration method to find optimal solutions for the problem by exploiting some fundamental properties of P that we have proved in [10]. The method avoids the intermediate storage of partial spanning trees in the course of the enumeration which is the major disadvantage of the approach of [12]. Instead, it makes use of a *bridge concept* devised by [7] for the non-redundant enumeration of spanning outtrees combined with the concept of so called *T-minimal trees* that has been proposed in [11].

Before we can describe our approach in detail we need to introduce some basic concepts devised by [9].

Let $O \subset V \times V$ be a *strict order* (i.e., an asymmetric and transitive binary relation) in activity set V .

$$\mathcal{S}_T(O) := \{S \in \mathcal{S}_T \mid S_j \geq S_i + p_i \ \forall (i, j) \in O\}$$

is called *order polytope* of O . For the empty strict order $O = \emptyset$ we have $\mathcal{S}_T(\emptyset) = \mathcal{S}_T$, and if $\mathcal{O}$ is the (finite) set of all inclusion-minimal feasible strict orders in activity set V , we have $\mathcal{S}_T = \bigcup_{O \in \mathcal{O}} \mathcal{S}_T(O)$. For problem P it can be shown that there is always an extreme point S^* of some order polytope $\mathcal{S}_T(O)$ which is a minimizer of f on $\mathcal{S}_T \neq \emptyset$. Moreover consider Network $N(O)$, which results from project network N by adding a precedence arc $\langle i, j \rangle$ with weight p_i for each pair $(i, j) \in O$. If N already contains an arc $\langle i, j \rangle$, its weight δ_{ij} is replaced by $\max(\delta_{ij}, p_i)$. Then each extreme point S^* of some order polytope $\mathcal{S}_T(O)$ can be represented by a spanning tree of order network $N(O)$, where the $n + 1$ arcs of this spanning tree T , say arcs $\langle i, j \rangle \in E^T$ with weights δ_{ij}^T , correspond to $n + 1$ linearly independent binding temporal constraints $S_j - S_i = \delta_{ij}^T$ ($\langle i, j \rangle \in E^T$). Together with $S_0 = 0$, the corresponding linear system of equations has

a unique solution, namely the vertex in question. The representation of extreme points by spanning trees is outlined in Figure 1. For the given project network $N(\emptyset)$ (cf. (a)) and order polytope $\mathcal{S}_T(\emptyset)$ (b) both spanning trees (c) represent the same extreme point $S = (0, 4, 4, 4, 6)$.

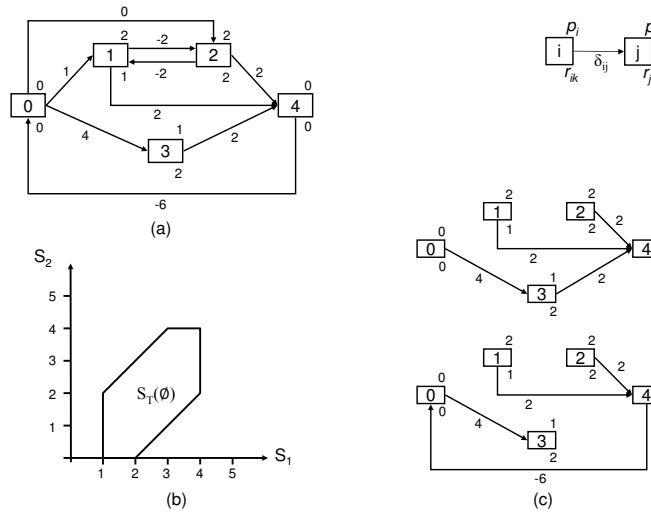


Fig. 1 Solutions represented by Spanning Trees

3 Tree-based enumeration approach

An optimal solution to problem P can be determined by consecutively fixing start times of activities such that, step by step, temporal constraints $S_j - S_i \geq \delta_{ij}^T$ become binding. We must ensure that the arcs, that correspond to these temporal constraints, constitute a spanning tree of some order network $N(O)$.

The basic idea of our approach is now to enumerate over all extreme points of order polytopes $\mathcal{S}_T(O)$. Consequently, we first construct an extended network $N(\hat{O})$, where $\hat{O}$ is the conjunction of all feasible strict orders $O \in \mathcal{O}$. Subsequently, we enumerate all time-feasible spanning trees for network $N(\hat{O})$ and finally, we determine the objective function value f evaluating the resource profiles $r_k(S, t)$ for all resources and the corresponding schedule S .

We first sketch how to generate all spanning trees of $N(\hat{O})$. This concept will then be improved to enumerate only time-feasible trees and finally we will apply the T-minimality-concept of [12] to avoid the generation of several spanning trees leading to one and the same schedule (cf. Figure 1).

3.1 Enumeration of spanning trees

Let $E^+ \supseteq E$ be the set of arcs of the extended project network $N(\hat{O})$. In what follows we call some tree T with $V^T = V$ and $E^T \subseteq E^+$ a spanning tree and some tree $T^\sim$ with $V^{T^\sim} \subset V$ and $E^{T^\sim} \subseteq E^+$ a partial tree of $N(\hat{O})$. $V^A = V \setminus V^{T^\sim}$ be the set of nodes that are currently not part of $T^\sim$. Finally, we call some arc $\langle x, v \rangle$ being part of the currently inspected tree $T^\sim$ *bridge* if there is no alternative arc that connects some node $i \in V^{T^\sim} \setminus \{v\}$ with any node $j \in V^A \cup \{v\}$. With regard to the last disconnected node v set V^A contains exactly the successors of v in the search tree of the last completed spanning tree. Figure 2 shows a case where arc $\langle x, v \rangle$ represents no bridge which is due to the dotted arcs connecting current tree $T^\sim$ with nodes from set V^A (Case (1)).

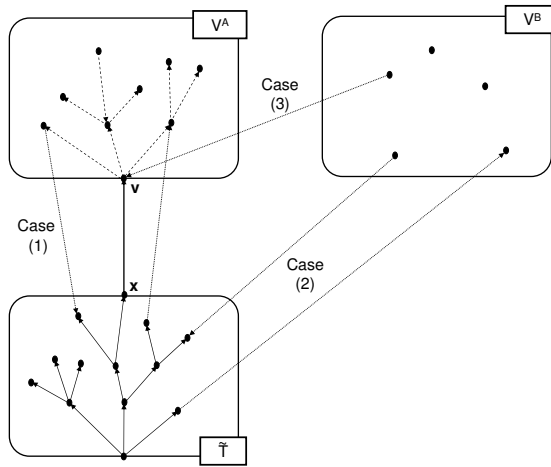


Fig. 2 Bridge-Test for spanning trees

Assume we consider some partial tree $T^\sim$. According to the concept of [7] we enumerate all spanning trees T with $E^T \supset E^{T^\sim}$, where the enumeration follows a depth first search strategy. Consequently, we expand a given partial tree in each iteration by a new arc leading from some node $i \in V^{T^\sim}$ to a node $j \in V^A$. Having enumerated all spanning trees containing partial tree $T^\sim$ we destroy $T^\sim$ by deleting the most recently added arc, say arc $\langle x, v \rangle$, and enumerate henceforth all trees containing arcs $E^{T^\sim} \setminus \langle x, v \rangle$. Following this concept we inductively find all spanning trees of the underlying order network $N(\hat{O})$. Each time we remove an arc from current tree $T^\sim$ we have to perform a so called *bridge-test*, which is crucial for the performance of the approach and will be detailed further on.

However, if some arc $\langle x, v \rangle$ that has been removed from $T^\sim$ turned out to be no bridge, we intermediately remove $\langle x, v \rangle$ from network $N(\hat{O})$ as well. Thus, we prevent to select this arc for the resulting partial tree again and with that to generate redundant trees. Subsequently we start re-growing the reduced partial tree via an alternative arc until we find the next spanning tree. If $\langle x, v \rangle$ represented a bridge we further reduce partial tree $T^\sim$ and restore all arcs that have been removed from network $N(\hat{O})$ within

the preceding iteration $|E^{T^-}|$. Intuitively, we may select these arcs (e.g. $\langle x, v \rangle$) once again, if we have modified $T^- \setminus \langle x, v \rangle$. The algorithm terminates if we are back in iteration $|E^{T^-}| = 0$ and the bridge-test shows the last removed arc to be a bridge.

Since we are not interested in spanning trees that lead to infeasible schedules, we restricted the extension of partial trees to generate only time-feasible spanning trees out of $N(\widehat{O})$. Since in a spanning tree each arc represents a binding temporal constraint (cf. Section 2) and not all combinations of binding time lags can be realized feasibly, we need to ensure that we do not add arcs to T^- that lead to infeasible start times for one or more activities. In particular, the set of precedence arcs $e^{pred} \in \widehat{O}$ that become part of a spanning tree must constitute a feasible strict order $O \in \widehat{O}$.

To ensure that we extend a spanning tree time-feasibly, we test in each iteration if one or more arcs become infeasible (i.e. lead to infeasible activity-start-times) when we add a new arc $\langle i, j \rangle$ to partial tree T^- . If that is the case, we temporarily lock the infeasible arcs until we remove $\langle i, j \rangle$ from T^- again. By this, we ensure that there are only feasible arcs selectable to extend T^- .

However, with this restriction we may come to a state where we cannot time-feasibly extend some partial tree. We call such an incomplete tree *pseudotree*. The general proceeding remains the same if we find a pseudotree. We only need to enhance the bridge-test with additional cases that need to be regarded if the last found tree has been a pseudotree.

Considering pseudotrees we need moreover to refine the definition of set $V^A = V \setminus V^{T^-}$. This is due to the fact, that we must now further distinguish between set V^A of nodes that are currently not part of partial tree T^- but have been part of the last pseudotree T and an additional set V^B of network nodes that have not been part of T . In this case $V^A \cup V^B = V \setminus V^{T^-}$ holds to be true. To decide whether arc $\langle x, v \rangle$ represents a bridge or not we must now test if one of the following conditions is satisfied.

- (1) There is a feasible arc $\langle i, j \rangle$ connecting some node $i \in V^{T^-}$ with some node $j \in V^A$
- (2) There is a feasible arc $\langle i, j \rangle$ connecting some node $i \in V^{T^-}$ with some node $j \in V^B$ and moreover there exists a sequence of arcs between node j and some node $l \in V^A$
- (3) There is a feasible arc $\langle j, l \rangle$ connecting some node $l \in V^B$ with some node $j \in V^A$ given that there is at least one arc that is not incident to the disconnected node v and that is a feasible extension in the following step

The three cases are outlined in Figure 2. If none of these cases is given $\langle x, v \rangle$ is a bridge and we proceed as described for general trees without time constraints.

3.2 Enumeration of Schedules

We can further reduce the set of candidate solutions to be examined since we will usually find more than one spanning tree of some network $N(O)$ leading to one and the same candidate schedule (cf. Figure 1). To avoid the redundant generation of schedules we applied the concept of *T-minimal trees* that was introduced by [11].

Let $\prec_a$ in E^+ be a strict order with $\langle i, j \rangle \prec_a \langle i', j' \rangle$ if either $i < i'$ or $i = i' \wedge j < j'$, ($\langle i, j \rangle, \langle i', j' \rangle \in E^+$). Then we must ensure that each node $j \in V$ is connected to a partial tree only by the minimum arc $\min_{\prec_a} \{\langle i, j \rangle | \langle i, j \rangle \in E^+\}$ leading to a certain activity start time S_j .

If we want to connect a new node v to the current tree T^- via some arc $e := \langle x, v \rangle$ or $e := \langle v, x \rangle$, respectively, in the course of our enumeration we have to ensure, that

there is no other arc, say $e' := \langle x', v \rangle$ or $e' := \langle v, x' \rangle$, respectively, that leads to the same start time S_v for node v and for which $e' \prec_a e$ ($e, e' \in E^+$) holds. Furthermore, if there is some arc e' with $e \prec_a e'$ leading to the same start time for the new activity e' must be the maximum arc (with respect to $\prec_a$) in the semi cycle that would arise if we added e and e' both to T^- . If one of these two conditions is violated the extension of T^- with arc e is said to be not T-minimal. Extensions that are not T-minimal need not to be investigated further.

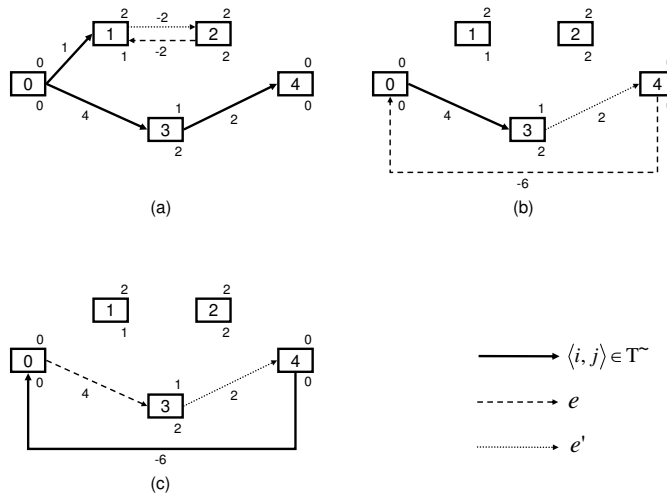


Fig. 3 T-minimal trees

Figure 3 illustrates the concept of T-minimal trees. For the project network from Figure 1(a) the trees depicted in (a), (b) and (c) are no feasible extensions. For case (a) the issue is obvious. We want to add activity 2 via arc $e := \langle 2, 1 \rangle$ to the current partial tree T^- . Since the reverse arc $e' := \langle 1, 2 \rangle$ leads to the same start time $S_2 = 3$ and $\langle 1, 2 \rangle \prec_a \langle 2, 1 \rangle$ this extension would violate the above concept of T-minimality. In extension (b) arc $e := \langle 4, 0 \rangle$ is examined to join T^- . This arc would imply start time $S_4 = 6$ for node 4. However, it leads not to a T-minimal tree because there is an alternative arc, namely $e' := \langle 3, 4 \rangle$, leading to the same start time for activity 4 with $\langle 3, 4 \rangle \prec_a \langle 4, 0 \rangle$. Thus, we dismiss this extension as well. The same is with the last example (c). If we connect node 3 to T^- via arc $e := \langle 0, 3 \rangle$ we find again an alternative arc leading to one and the same start time for this node, namely arc $\langle 3, 4 \rangle$. Comparing the two arcs directly will show the currently intended extension arc to be smaller than the alternative. If we investigate the arising semi cycle $0 \rightarrow 3 \rightarrow 4 \rightarrow 0$ however, we find out that the maximum arc $\langle 4, 0 \rangle$ is part of T^- and this extension is not T-minimal either.

3.3 Branch and Bound

We embedded our enumeration scheme into a branch and bound procedure to avoid the examination of solutions that can be recognized as non-optimal early in the construction of a spanning tree. Thus, we compute a *lower bound* for the objective function value of each partial tree $T^\sim$ in our enumeration to verify that we can construct a spanning tree out of $T^\sim$ leading to a better solution as the *upper bound* which is the current best solution. We decided to initialize the upper bound of our procedure with the first feasible solution of our enumeration since we generally receive this first solution after a negligible amount of time.

We proposed an appropriate lower bound for our objective function in [6]. A lower bound for some partial tree $T^\sim$ can be achieved by generating a resource profile based on the fixed resource demands implied by partial schedule $S^{T^\sim}$ and on anticipations of the remaining resource demands of activities that are not yet part of $T^\sim$. The construction of this resource profile follows four steps and is illustrated in Figure 4.

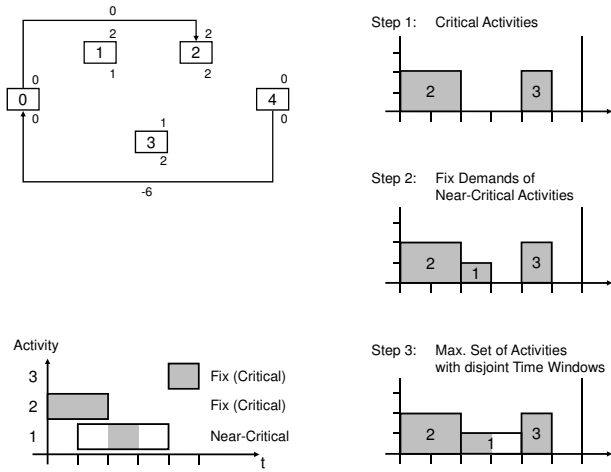


Fig. 4 Construction of a Lower Bound for the RLP

- (1) In a first step we initialize the resource profile as indicated with the resource demands of all activities $i \in V^{T^\sim}$, for which we already have fixed the start time S_i
- (2) In a second step we add the fix resource demands of all critical and near-critical activities $j \in V \setminus V^{T^\sim}$ to the time interval between the activity's latest possible start time LS_i up to its earliest completion time EC_i
- (3) In the third step we add the demands of a maximum set of activities $l \in V \setminus V^{T^\sim}$ with disjoint time windows as best as possible to the resource profile
- (4) Finally, we strip down all remaining activity resource demands in a set of independent 1-resource-unit-for-1-time-unit demands. These 1-bundles are distributed as best as possible between $\min\{ES_m | m \in V \setminus V^{T^\sim}\}$ and $\max\{LC_n | n \in V \setminus V^{T^\sim}\}$

For the partial tree shown in Figure 4 only steps 1-3 of our concept had to be applied because after we had added all demands of the fixed activities 2 and 3 as well as the critical demands of near-critical activity 1 to our resource profile in Step 1 and Step 2 we had only near-critical activity 1 left in Step 3 to form a maximum set of activities with disjoint time windows. Thus, we add the remaining resource demands of activity 1 as best as possible according to the advises for Step 3 and the resource profile for our lower bound is completed.

For a resource profile that has been built up according to this concept we can now easily determine the function value that will serve as lower bound for the partial tree under consideration. If we consider more than one resource we build up one resource profile for each resource. The corresponding function values can simply be summed up to the overall objective function value (cf. [6]).

If the lower bound value of some partial tree T^* turns out to be already worse than the current upper bound we can cancel the examination of this partial tree and destroy it by the deletion of the last arc just as we would do in the case that there was no selectable arc to expand the current tree.

4 Computational study

To verify the performance of our approach we compared it to our time-window-based start time enumeration approach [6] and to a tree-based approach presented in [12] for the resource renting problem which we adapted to problem P . All approaches were embedded in a branch and bound procedure using the lower bound that has been described in Section 3.3.

The tests have been performed using an *Intel Core 2 Duo T9300 with 2,5 Ghz clock pulse and 3,5Gb RAM*. A test has been canceled after a deadline of 25.000 seconds computation time.

For our tests we used different test sets. First of all we used the well known *rlp-j10* test set devised by [13]. This test set contains 270 test instances with 10 activities and 1-5 resources. Furthermore, we will present the results of the tree-based methods on the *rlp-Ic* test set devised by [6] which contains 90 test instances (instances 91-180) with 15 activities and 1-5 resources and finally on 45 instances of the *rlp-j20* test set (cf. [13]) with 20 activities and 1-5 resources.

On the *rlp-j10* test set we tested all methods several times. Ignoring the prescribed deadlines of the test instances we started for each instance with the tightest possible deadline $\bar{d} = ES_{n+1}$ and then added an overall project buffer to the deadline which implies growing time windows for most activities. In table 1 we can see the average computation time $CT_{\emptyset}$ per test instance as well as the number of solved instances in 1 and 10 seconds, respectively, for this test set with $\bar{d} = ES_{n+1}$ and $\bar{d} = (ES_{n+1} \cdot 1.25)$.

Project Buffer	0%			25%		
	$CT_{\emptyset}$	# < 1s	# < 10s	$CT_{\emptyset}$	# < 1s	# < 10s
Our Approach	0,15s	262	270	0,71 s	230	268
Nübel Adaption	0,29s	234	269	1,49 s	141	260
Time Window	1,13s	238	260	1.671,31s	59	156

Table 1 Computational results: rlp-j10 test set

Due to its inferior performance on the *rlp-j10* test set the time-window approach has not been further evaluated. The results of the tree-based methods on the test sets *rlp-Ic* and *rlp-j20* can be found in tables 2 and 3.

Project Buffer	0%				
	$CT_{\emptyset}$	$\# < 1s$	$\# < 10s$	$\# < 1000s$	$\#_{opt}$
Our Approach	62,71s	26	53	89	90
Nübel Adaption	169,36s	13	46	88	90

Table 2 Computational results: rlp-Ic test set (Instances 91-180)

Project Buffer	0%				
	$CT_{\emptyset}$	$\# < 1s$	$\# < 10s$	$\# < 1000s$	$\#_{opt}$
Our Approach	2.285,34s	3	11	25	36
Nübel Adaption	3.535,82s	0	11	23	34

Table 3 Computational results: rlp-j20 test set (Instances 136-180)

On the *rlp-j10* test set we find that the tree based methods perform superior by far. Furthermore, the sensibility to slackened project deadlines is generally far less distinctive. Comparing our approach against the enumeration scheme of [11] we observed that our approach performed significantly better on all test sets.

References

1. Ahuja H.N., Construction Performance Control by Networks. John Wiley and Sons, New York (1976)
2. Ahuja R.K., Magnanti T.L., Orlin J.B., Network Flows. Prentice Hall, Englewood Cliffs (1993)
3. Ballestin F., Schwindt C., Zimmermann J., Resource leveling in Make-to-Order Production: Modeling and Heuristic solution Method, International Journal of Operations Research, 4, 50–62 (2007)
4. Bandelloni M., Tucci M., Rinaldi R., Optimal Resource Leveling using non-serial dynamic programming, European Journal of Operations Research, 78, 162–178 (1994)
5. Easa S.M., Resource Leveling in Construction by Optimization, Journal of Construction Engineering and Management, 115 2, 302–316 (1989)
6. Engelhardt H., Zimmermann J., Lower Bounds and Exact Methods for Resource Levelling Problems, Technical Report WIOR (Universität Karlsruhe), 517 (1998)
7. Gabow H.N., Myers E.W., Finding all spanning trees of directed and undirected graphs, Siam Journal on Computing, 7, 280–287 (1978)
8. Kolisch R., Schwindt C., Sprecher A., Benchmark Instances for Project Scheduling Problems. In: J. Węglarz (Hg.) Project Scheduling Recent Models, Algorithms and Applications, 197-212. Kluwer, Dordrecht (1999)
9. Neumann K., Nübel H., Schwindt C., Active and stable project scheduling, Mathematical Methods of Operations Research, 52, 441–465 (2000)
10. Neumann K., Schwindt C., Zimmermann J., Project Scheduling with Time Windows and Scarce Resources. Springer, Berlin (2003)
11. Nübel H., Minimierung der Ressourcenkosten für Projekte mit planungsabhängigen Zeitfenstern. Gabler, Wiesbaden (1999)
12. Nübel H., The resource renting problem subject to temporal constraints, OR Spektrum, 23, 359–381 (2001)
13. Węglarz J., Project Scheduling: Recent Models, Algorithms, and Applications. Kluwer, Boston (1999)

Using a set of fixed prices when solving Profit Maximization Capacitated Lot-Size Problems (PCLSP)

Arne Løkketangen · Kjetil Haugen ·
Guillaume Lanquepin · Asmund Olstad

1 Introduction

The Capacitated Lot-Size Problem, CLSP, is central in production planning research, and a lot of publications on this problem exists (see e.g. [2]). Many formulations, variants and extensions exist for this problem. We have previously looked at this problem using a profit maximization formulation instead of the customary cost minimization (see [4]). This work has later been extended by solving larger instances, using heuristics, and adding price limits (see [5]). The reasoning behind these extensions is that instead of having a fixed demand, the actual demand can be influenced by changing the prices for these commodities. This again influences the required capacity. As noted in our previous work, we assume a monopolistic situation. This is somewhat unrealistic, but we argue that our work might be transferable to more realistic (e.g. oligopolistic/game-theoretic) market environments. The main point is that demand can be affected, not necessarily the complexity of the demand description.

In [3] the authors argue that there are three factors driving the trend towards demand based management: (1) increased availability of demand data, (2) it is becoming

Arne Løkketangen
Molde University College
Box 2110, 6402 Molde, Norway
E-mail: Arne.Lokketangen@himolde.no

Kjetil Haugen
Molde University College
Box 2110, 6402 Molde, Norway
E-mail: Kjetil.Haugen@himolde.no

Guillaume Lanquepin
Molde University College
Box 2110, 6402 Molde, Norway
E-mail: Guillaume.Lanquepin@hiMolde.no

Asmund Olstad
Molde University College
Box 2110, 6402 Molde, Norway
E-mail: Asmund.Olstad@himolde.no

increasingly easier to change the prices, and (3) the increased availability of decision support tools. They claim that dynamic pricing will migrate from the hotel and airline industry to retail and other industries where sellers are able to store inventory. Lee [9] writes:

What will be the next competitive battleground in the 21st century? It will be managing the demand for the total value maximization of the enterprise and value chains - Demand- Based management.

In our previous work the prices were continuous, and could take on any value within its bounds. This is a somewhat unrealistic condition. It is much more common for most commodities that the prices are discretized, with a certain distance between the price levels. We will look at this case of a fixed set of prices per commodity. Of interest are the consequences of the discretization algorithmically and how the computational results are affected. We will also look at the consequences of the various pricing policies that can be applied.

This is work-in-progress. Novel heuristics, based on Tabu Search (see [6]), but also integrating Cplex [1], and classical heuristics based on the Wagner-Whitin [10] and Thomas [8] algorithms as subproblem solvers will be developed. Extensive computational results will be reported, based on (modified) instances from Lotsize-lib [7] and on instances previously published by the authors.

References

1. <http://www.ilog.com/products/cplex/>
2. A. Drexler and A. Kimms, Lot Sizing and scheduling - survey and extensions, European Journal of Operational Research, Vol 99, pp 221 - 235 (1997)
3. W. Elmaraghraby and P. Keskinocak. Dynamic pricing in the presence of inventory considerations: Research overview, current practices, and future directions., Management Science, Vol 49 (10), pp 1287 - 1309 (2003).
4. K. K. Haugen, A. Olstad and B. I. Pettersen, Solving Large-scale Profit Maximization Capacitated Lot-Size Problems by Heuristic Methods, Journal of Mathematical Modelling and Algorithms, Vol. 6, pp. 135 - 149, (2007).
5. K. K. Haugen, A. Olstad and B. I. Pettersen, The profit maximizing capacitated lot-size (PCLSP) problem, European Journal of Operational Research, Vol 176, pp 165 - 176 (2007).
6. F. Glover and M. Laguna, Tabu Search, Springer, (1996).
7. <http://www.core.ucl.ac.be/wolsey/lotsizel.htm>
8. L.J. Thomas, Price-production decisions with deterministic demand, Management Science, Vol 18 (11), pp 747 - 750 (1970).
9. H.L. Lee, Ultimate Enterprise value creation using demand-based management, Working Paper SGSCMF-W1-2001, Stanford University (2001).
10. H.M. Wagner and T.M. Whitin, Dynamic version of the economic lot size model, Management Science 5 (3), pp 89 - 96 (1958).

An Investigation of an Ant-based Hyperheuristic for the Capacitated Vehicle Routing Problem

Zalilah Abd Aziz • Graham Kendall

1 Introduction

One of the aims of hyperheuristics is to develop more general systems that are able to solve a wider range of problems than is currently possible with more conventional search methodologies. Hyperheuristics can be defined as “*heuristics to choose heuristics*” (Burke et al. 2003), with their strength being in their ability to combine a number of simple heuristics to search for good solutions. These simple heuristics can be, for example, k -opt moves or simple local search moves that are problem dependent used by problem experts to construct solutions.

In this research, we intend to investigate ant-based hyperheuristic which are capable of managing a set of low-level heuristics utilising an ant algorithm (originally proposed by Dorigo et al. (1996)). A similar approach has previously been applied to two optimisation problems; the project presentation scheduling (Burke et al. 2005) and the travelling tournament problem (Chen et al. 2007). This research will extend these initial investigations and explore the effectiveness of the algorithm using the Capacitated Vehicle Routing Problem (CVRP).

2 Ant-based Hyperheuristics

In the proposed ant-based hyperheuristic, a problem is represented as a directed graph where the nodes represent low level heuristics. Initially, ants traverse the graph, carrying with them the initial solutions. To decide which heuristic (node) to visit next, the ants make the decision probabilistically.

The proposed ant-based hyperheuristic incorporates the following features:

1. *How ants make the decision as to which heuristic (node) to visit next?* The decision will be made based on the probability transition rule as defined in the Ant System (Dorigo et al. 1996), but suitably adjusted for a hyperheuristic framework.
2. *Pheromone value?* As the ants travel, they deposit a chemical substance called pheromone. In a standard Ant System, the amount of pheromone corresponds to the quality of the solution found by the ant. In this respect the hyperheuristic algorithm mirrors the original ant algorithm.
3. *Visibility?* Visibility information represents some form of heuristic information, which is combined with the pheromone value in order to decide which node (heuristic) to visit next. In the original ant algorithm, the heuristic information was the distance between two cities (as it was tackling the traveling salesman problem). As such, it remained static throughout the algorithm. In our proposed algorithm the heuristic information is how well two heuristics work together. This is measured as the computational time and, as such, it is now a dynamic value over the course of the algorithm.

3. Problem Description : Capacitated Vehicle Routing Problem (CVRP)

The Capacitated Vehicle Routing Problem (CVRP) is a version of the Vehicle Routing Problem (VRP) where a fleet of m vehicles with limited capacity Q has to visit a set of customers with specific demands. The sum of demands on any route should not exceed the vehicle capacity Q . The objective is to minimize the total travel distance. The CVRP can be formulated as follows: a connected graph $G = (V, E)$ of $n+1$ nodes, with a set of customers with demand q_i is represented as $v_i \in V, i = 1, \dots, n$ with v_0 as the depot.

The CVRP problem is to find a set of least cost vehicle routes so that the following constraints are satisfied:

1. Each vertex except the first vertex (depot), v_0 , is served exactly once by exactly one vehicle.
2. Each vehicle's route starts and ends at the depot.
3. The total demand does not exceed the vehicle capacity Q for every route.

4 Low Level Heuristics

Low level heuristics are held in the nodes of the graph. Usually, they represent simple local search neighborhoods such as swap, move or rules that were used by the user to construct the solutions. In the proposed hyperheuristic framework, low level heuristics will generate a new solution and we will maintain other information such as changes in the evaluation function and computational time taken to carry out the heuristic.

We have drawn from the literature (Braysy and Gendreau 2005), a set of low level heuristics for the CVRP. They are briefly described here:

1. **2-opt exchange operator** – replace two edges by two other edges and iterates until no further improvement is possible.
2. **Or-Opt operator** – replace three edges in the original tour by three new ones without modifying the orientation of the new route.
3. **2-opt* Operator** – combine two routes so that the last customers of a given route are introduced after the first customers of another route.
4. **Relocate Operator** – moves a random customer from its current route to a different one.
5. **Exchange Operator** – swaps two customers in different routes.
6. **Ejection-chain** – moves a customer from its current route and insert it into another route.
7. **Geni-Exchange Operator** – a customer is inserted between two customers on the nearest destination route.

5 Experimental Setup

In order to provide a sound foundation for our ant-based hyperheuristic, we have reproduced the original ant algorithm (Ant System). We have tested the algorithm on the Travelling Salesman Problem (TSP) and shown that it is not statistically significantly different from the original algorithm.

5.1 Ant-based Hyperheuristic

Our proposed algorithm is based on the ant algorithm of Dorigo et al. (1996) but suitably changed to cater for the dynamic heuristic information (i.e. CPU time of two heuristics) rather than the static information (distance between two cities) for the TSP implementation. The ant-based hyperheuristic is outlined in the following pseudocode:

1. $t = 0$ *t is the journey counter. A journey is equivalent to the number of low level heuristics available in the algorithm.*
2. $counter = 0$ *user defined counter for how many cycles ant will make a journey*
3. $\tau_{ij}(t) = c; \Delta \tau_{ij} = 0$ *c is an initial value of pheromone value given for every edge (i,j)*
4. $\eta_{ij} = 1$ *visibility is initially given value 1*
5. Set S_k and S_b as null *S_k is each ant's solution, S_b is best solution*
6. Place m ants on n heuristics
7. For $k = 1$ to m *place the starting node of the ants.*
 Repeat until ants complete a journey
 For $k = 1$ to m do
 Select the next heuristic (node j) to apply based on pheromone and visibility
 Move the k -th ant to heuristic j
 Apply heuristic j to S_k to obtain new solution S'_k
 If ($S'_k < S_b$)
 $S_b = S'_k$
 Update visibility η_{ij} *visibility is updated after a heuristic is applied*
 $t = t + 1$
 end for
 $counter++$
8. If ($counter == n$) *a journey is completed*

```

    Update pheromone value  $\tau_{ij}$ 
    counter = 0
    If ( any best solution is in this cycle)
        For all ants  $S_k = S_b$ 
    Endif
Endif
9. If (  $t = t_{max}$ )
    Output best solution  $S_b$  found
Stop

```

We will discuss the algorithm in more detail during the conference presentation.

5.2 Initial Solutions

The initial solutions for the CVRP have been generated using the sweep algorithm (Gillet and Miller 1974). The algorithm is briefly described as follows:

- Step 1: The polar coordinates of each node are calculated and sorted in increasing order.
- Step 2: Starting from an initial node, choose the next node that has the smallest angle. Assign that node to vehicle route k . The nodes are clustered such that the total capacity of each route does not exceed the capacity Q .

6. Future Work

We are in the process of adapting the original ant algorithm in order to produce a hyperheuristic version. Following this we will implement the heuristics presented in section 4. This will enable us to run a series of experiments to test the effectiveness of the proposed algorithm. At the conference we will present the full algorithmic details as well as the results when run on benchmark CVRP instances.

References

1. Braysy O. and Gendreau M. *Vehicle Routing Problem with Time Windows, Part 1: Route Construction and Local Search Algorithms*, Transportation Science, 39, 104-118 (2005).
2. Burke E.K., Hart E, Kendall, G. Newall, J., Ross P. and Schulenburg S. *Hyper-Heuristics: An Emerging Direction in Modern Search Technology*. Handbook of Meta-Heuristics, F. Glover and G. Kochenberger (Eds), Kluwer Academic Publishers, 457-474 (2003).
3. Burke E. K., Kendall G., Landa-Silva J. D., O'Brien R. and Soubeiga E. *An Ant Algorithm Hyperheuristic for the Project Presentation Scheduling Problem*, In Proceedings of the 2005 IEEE Congress on Evolutionary Computation, 3, 2263-2270 (2005).
4. Chen P., Kendall G. and Vanden-b G. *An Ant Based Hyper-heuristic for the Travelling Tournament Problem*. In Proceedings of IEEE Symposium of Computational Intelligence in Scheduling (CISched 2007), 19-26 (2007).
5. Dorigo M., Maniezzo V. and Colomi A. *Ant System: Optimisation by a Colony of Cooperating Agents*. IEEE Transactions on Systems, Man and Cybernetics, 26, 29-41(1996).
6. Gillet B.E. and Miller L.R. *A Heuristic Algorithm for the Vehicle Dispatch Problem*. Operations Research, 22, 340-349 (1974).

Solving the Undirected Capacitated Arc Routing Problem with a memory based Method

Istvan Borgulya

1 Introduction

In the capacitated arc routing problem (CARP) there is a network of streets, and on a subset of streets we have to make services (e.g. winter gritting, litter collection, school bus scheduling). A fleet of identical vehicles of limited capacity provides services; the fleet is based at a depot and each trip starts and ends at the depot. Each street can be traversed any number of times, with a known cost. The CARP consists of determining a set of vehicle trips of minimum total cost, such that each required street is serviced by one single trip, and the total demand processed by a trip fits vehicle capacity.

The CARP is NP-hard. Exact methods for the CARP (like the branch and bound method) have only been able to solve relatively small instances to optimality. For larger instances different heuristic algorithms have been proposed. Commonly used heuristics are the constructive heuristics e.g. the Path-scanning, Ulusoy's tour splitting algorithm. The metaheuristics are also important: e.g. taboo search, evolutionary algorithm (EA) and variable neighbourhood search.

In this paper we show a new EA for the undirected CARP. This EA is a hybrid steady-state algorithm. It works without recombination operator and it uses a memory based technique, the extended virtual loser (EVL) [1] for mutation. To modulate these methods for the CARP: we adapted the EVL for the CARP, we used a restart strategy in the EVL, we used two special versions of the mutation operator and we used five special stochastic 2-opt local search algorithms to improve the solutions.

The published evolutionary algorithms (e.g. [3], [4]) solve the CARP problems by using problem specific, sophisticated operators. We will show that our approach is a simpler one.

2 The new algorithm

The new algorithm called CARP_E (CARP with EVL technique) uses a two-stage algorithm structure in order to produce higher quality results. Each stage is a steady-state EA. The first stage is a quick “preparatory” stage which is designated to improve the quality of the

Istvan Borgulya

University of Pecs, Faculty of Business and Economics, Hungary,

E-mail: borgulya@ktk.pte.hu

initial population. The second stage is a usual steady-state EA. In both stages the algorithm improves the accuracy of the solutions with local search procedures.

Let us see first the main steps of the new algorithm:

Procedure CARP_E($t, itt, kn, itend$)

Initial population. Initial values of ECM. $it=0$

/* First stage

Do itt times

$it = it + 1$. A descendant is generated randomly. Local search. Reinsertion.

In every kn th generation: Update of ECM, deleting the duplicate individuals.

od.

/* Second stage

Repeat

$it = it + 1$. Truncation selection, mutation based on EVL, local search.

Reinsertion.

In every kn th generation: Update of ECM, deleting the duplicate individuals. Restart.

until $it > itend$

end

The parameters of the algorithm: t - the size of the population, itt - the number of the generations in the first stage, kn - the algorithm is controlled in every kn th generation, $itend$ - the maximum number of the generations (ECM: memory of the EVL technique).

The operations and the characteristics are the following:

Route notation. Our network is a directed graph using two opposite arcs per edge. An arc index list is used. If u is the index of an arc, the index of the opposite arc is $1000 + u$. Each trip is stored as a sequence of required arcs with their index. The shortest path between the required arcs is not stored.

Individuals. The individual describes m vehicle trips. This is a permutation of required arc's indices and the depot several times. The index of the depot is null, and every trip begins with null. (Cyclic permutations are considered identical.)

Initial population. One part of the population is generated randomly; the other part (e.g. a third of the individuals) is generated based on a nearest neighbour lists.

Constraints. Each trip belongs to a vehicle and the trip demand is at most C . The trip demands are controlled by constraints: the i th trip has a constraint:

$$g_i : \text{trip demand of the } i\text{th trip} - C \leq 0 \quad (i = 1, 2, \dots, m)$$

Fitness function. The fitness function gives a relative order between the individuals based on the objective function of the problem and the constraints.

Mutation. We use randomly two different mutation versions. The first version is the mutation based on the EVL [1], and the second is two randomly chosen moves: an arc inversion (opposite arc), or a swap between sub trips from two selected different trips.

Local-search. In the algorithm we apply five versions of the 2-opt local search algorithm examining different moves. The examined moves of two selected arcs are the following: invert all arcs between the two required arcs, swap the two arcs, insert the second arc after the first arc, swap the ending sub trips from the selected arcs to the depot if the arcs are parts of different trips, swap the beginning sub trips from the depot until the selected arcs if the arcs are parts of different trips. Every local search is stochastic: if they cannot improve the solution, they accept the wrong solution with a small probability. Our algorithm applies the sequence of the local search procedures for each new solution.

Reinsertion. In all stages the algorithm compares the descendant with the most similar former individual (based on the Hamming distance between the individuals). If the descendent is better than the former solution, it is replaced by the descendent. If the number of the individuals is fewer than the size of the population, the algorithm inserts the descendant (e.g. after restart).

Stopping criteria. The algorithm is terminated if a pre-determined number of generations have been performed.

3 Experimental results

Test problems, parameters

We used the well known *gdb*, *val* and *egl* files (<http://www.uv.es/~belengue/carp.html>).

Every test problem was run twenty times. The used parameters were the following: $t = 90$, $itt = 50$, $kn = 10$. The maximum number of the generations was between 10000 and 20000 depending on the problem. The CARP_E was implemented in Visual Basic and ran on Intel Core Duo CPU 2.2 GHz with 2 GB RAM under Windows Vista Business.

Results of CARP_E

Table 1 shows the average results. It shows problem name (*name*), the average deviation (%) from the lower bound of the best (*best*), the worst (*worst*) and the average (*aver.*) solutions; the standard deviation of the solutions (*SD*) and the average time to the best solutions (*time*) (CPU time in seconds).

Table 1. The average results of CARP_E.

name	best	worst	aver.	SD	time
<i>gdb</i> file	0.13	0.38	0.33	0.26	28.5
<i>val</i> file	0.29	0.52	0.40	0.41	227.5
<i>egl</i> file	1.59	2.48	2.00	34.69	5503.0

In the instance *egl-s4-B*, CARP_E found a new best known value: **16240** (the earlier was 16321). In this instance we let CARP_E to solve the problem with fewer vehicles: we got an even better solution again. The new best known value is **15938**.

Comparative results

To compare the results of our algorithm we chose four meta-heuristics. These include the variable neighbourhood search of Polacek et al. [5] (VNS), the ant colony approach of Lacomme et al. [4] (BACO), the memetic algorithm of Lacomme et al. [3] (LPR_MA) and the taboo search procedure of Brandão and Engleze [2] (BE_TS). The different CPU times were scaled to 2.2 GHz.

Table 2 presents the results of the comparison. The table exhibits the method name (*method*), the average deviation from lower bound (LB) for the best and the average solutions of the *gdb*, *val* and *egl* instances, and the average CPU time by the *gdb*, *val* and *egl* instances.

We compared the average results. Analyzing the results we can confirm that our CARP_E is the fourth method after VNS, BE_TS and LPR_MA based on the quality. If we compare the CARP_E with the other EAs (BACO and LPR_MA) its results are similar to the results of LPR_MA. But the LPR_MA method is 10-20 times faster than our CARP_E in general.

To improve the running time and to reach a faster convergence, we try to improve the algorithm in the future. We will analyze the local search technique and based on the running statistics of the local searches, we will reduce the number of the iterations in the local search procedures.

4 Summary

In this paper, we present a new evolutionary algorithm for the CARP. The algorithm is a hybrid method, uses several local search procedures and for the mutation it uses a memory based technique, the extended virtual loser (called EVL). The paper shows that our algorithm without problem specific, sophisticated selection, recombination and mutation operators can solve the CARP in a simple way. The results show that our algorithm has good quality and our algorithm found a new best solution for a difficult problem too.

Future efforts can focus on improving the results and the speed of the algorithm. So we plan to develop an island EA model for this problem and we try to improve the local search technique.

Table 2. Comparative results with different methods

Method	Average deviation from LB (%)						Average time		
	gdb		val		egl		gdb	val	egl
	best	aver.	best	aver.	best	aver.			
BACO	0.30	0.36	0.90	-	4.11	-	9.14	102.34	867.14
CARP_E	0.13	0.33	0.29	0.40	1.59	2.00	28.50	227.57	5503.21
LPR_MA	0.13	0.15	0.18	0.23	1.32	1.74	2.18	16.00	219.62
BE_TS	0.13	0.14	0.12	0.15	1.03	1.54	1.56	12.62	182.12
VNS	-	-	0.13	0.23	0.86	1.35	-	71.55	820.24

Acknowledgements The Hungarian Research Foundation OTKA K 68137 supported the study.

References

- [1] I. Borgulya: An Algorithm for the Capacitated Vehicle Routing Problem with Route Balancing *Central European Journal of Operations Research*. Vol. 16 No. 4. pp. 331-344 (2008)
- [2] J. Brandão and R. Eglese: A deterministic tabu search algorithm for the capacitated arc routing problem *Computers & Operations Research* 35. pp. 1112 – 1126 (2008)
- [3] P. Lacomme, C. Prins, W. Ramdane-Cherif: Competitive memetic algorithms for arc routing problems. *Annals of Operational Research* 131(1–4):159–85. (2004)
- [4] P. Lacomme, C. Prins and A. Tanguy: First competitive ant colony scheme for the CARP. Research report LIMOS/RR-04-21; [<http://www.isima.fr/limos/publi/RR-04-21.pdf>] (2004)
- [5] M. Polacek, K.F. Doerner, R.F. Hartl, V. Maniezzo: A variable neighbourhood search for the capacitated arc routing problem with intermediate facilities *J. Heuristics* 14: 405-423. (2008)

An Effective Heuristic for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading and Multiple Stacks

Michel Gendreau · Jean-François Côté ·
Jean-Yves Potvin

1 Introduction

In many practical contexts, loading constraints have a strong impact on the way in which delivery routes can be organized. This important dimension of routing operations has been realized since a few years and several authors have recently addressed variants of classical routing problems in which loading constraints are accounted for. In this presentation, we consider a routing problem in which a single vehicle must move a given number of objects from a set of pickup locations to a set of delivery locations subject to complex loading constraints. Each object is loaded at a specified pickup point and must be delivered to a specified delivery location. Objects are assumed to have a standard width and can thus be placed in a given number of “stacks” in the vehicle. Once loaded, objects cannot be moved to other stacks or resequenced in their stack, and only the (current) last object of each stack is available for delivery. Another constraint is that the total length of the objects making up each stack at any time must not exceed the length of the vehicle.

This problem is an extension of the one in which items are packed in a LIFO fashion on a single vehicle that performs pickup and delivery operations. This problem has been addressed recently by Carrabs et al. [1], who proposed a Variable Neighborhood Search approach for it, and in [2], where an exact method based on branch-and-bound was presented. The use of multiple stacks to model two-dimensional loading constraints on

Michel Gendreau
Interuniversity Research Centre on Enterprise Networks,
Logistics and Transportation (CIRRELT) and
Département d’informatique et de recherche opérationnelle (DIRO)
Université de Montréal
E-mail: michel.gendreau@cirrelt.ca

Jean-François Côté
CIRRELT and DIRO
Université de Montréal
E-mail: cotejean@iro.umontreal.ca

Jean-Yves Potvin
CIRRELT and DIRO
Université de Montréal
E-mail: potvin@iro.umontreal.ca

vehicles was introduced by Petersen [3] who considered a problem in which a vehicle must first pickup a group of unit-width objects and load them in several stacks, travel a long distance, and then deliver these objects subject to loading constraints. This problem differs significantly from ours in the fact that all pickup operations are grouped together at the beginning of the route, while we allow mixing pickup and delivery operations.

2 An “Adaptive Large Neighborhood Search” Heuristic

The solution method that we propose is based on the “Adaptive Large Neighborhood Search” heuristic that Pisinger and Ropke [4] applied very successfully to a large range of routing problems, including pickup and delivery ones. Pisinger and Ropke’s approach consists in systematically deleting a large fraction of the elements of the current solution using adapted destruction operators and then “reconstructing” a new solution from the remainder using adapted reconstruction operators. The selection of the specific destruction and construction operators to be applied in any given iteration is made randomly and depends on probabilities that are adjusted dynamically through a learning mechanism to reflect the good or bad performance of the various operators.

Our heuristic is made up of three phases. An initial solution is first constructed by sequentially inserting requests in the route and the stacks. Then, the main “Adaptive Large Neighborhood Search” phase is performed. It follows the procedure presented in [4], but uses four destruction operators that respectively remove randomly a given number of requests, a group of geographically close requests, a group of requests picked up sequentially, and the requests assigned to a given stack; two insertion operators, based respectively on minimum cost and minimum regret criteria, are applied to reconstruct complete solutions. It should be noted that the new solution derived at a given iteration replaces the current one only if it is an improving one or if it passes a probabilistic test as in simulated annealing. A third phase exploits the fact that a solution to the problem is defined both by the assignment of pickup and delivery points to stacks and by the exact route followed by the vehicle. For a given feasible set of assignments of pickup and delivery points to stacks, one may derive several feasible routes of different length. Finding the optimal route for a given assignment can be performed by a dynamic programming procedure of complexity $O(n^k)$, where n is the number of requests to be transported and k is the number of stacks in the vehicle.

3 Computational Results

The proposed procedure was tested on four sets of benchmark instances. The two first sets are for the single-stack variant of the problem and were proposed in [1] and [2]. The first set is made up of 42 instances having between 12 and 375 requests, while the second group of 63 instances contains from 9 to 21 requests. The third benchmark is the one proposed by Petersen [3]. It is made up 60 instances of the (DTSPMS) having a 3-stack vehicle and 12, 33 or 66 requests. DTSPMS instances can easily be transformed into instances of our problem by suitably defining distances between the depots and pickup and delivery locations. Finally, the fourth benchmark is a new set of instances based of the 28 first instances of [1]; three new instances having between 2 to 5 stacks

are defined for each original one and allow testing different features of the solution procedure.

References

1. F. Carrabs, J.-F. Cordeau, and G. Laporte, Variable Neighbourhood Search for the Pickup and Delivery Traveling Salesman Problem with LIFO Loading, *INFORMS Journal on Computing*, 19, 618-632 (2007)
2. F. Carrabs, R. Cerulli, and J.-F. Cordeau, An Additive Branch-and-Bound Algorithm for the Pickup and Delivery Traveling Salesman Problem with LIFO or FIFO Loading, *INFOR*, 45, 223-238 (2007)
3. H. L. Petersen, Heuristic Solution Approaches to the Double TSP with Multiple Stacks, Technical Report 2006-2, Centre for Traffic and Transport, Technical University of Denmark, Lyngby, Denmark (2006)
4. D. Pisinger and S. Ropke, A general heuristic for vehicle routing problems, *Computers & Operations Research*, 34, 2403-2435 (2007)

Variable Neighborhood Descent methods for the Distributed Permutation Flowshop Problem

Rubén Ruiz • Bahman Naderi

1 Introduction

In the permutation flowshop problem (PFSP) a set N of n unrelated jobs are to be processed on a set M of m machines. These machines are disposed in series and each job has to visit all of them in the same order. This order can be assumed to be $1, \dots, m$ without loss of generality. Therefore, each job $j, j \in N$ is composed of m tasks. Once a processing order for the first machine has been determined, this order is to be maintained throughout all machines. There is a common and universal assumption in the PFSP literature: there is only one production center, factory or shop. This means that all jobs are assumed to be processed in the same factory. Nowadays, single factory firms are less and less common, with multi-plant companies and supply chains taking a more important role in practice (Moon et al., 2002). In the literature we find several studies where distributed manufacturing enables companies to achieve higher product quality, lower production costs and lower management risks (see Wang, 1997 and Kahn et al., 2004 for some examples). However, such studies are more on the economic part and distributed finite capacity scheduling is seldom tackled. In comparison with traditional single-factory production scheduling problems, scheduling in distributed systems is more complicated. In single-factory problems, the problem is to schedule the jobs on a set of machines, while in the distributed problem an important additional decision arises: the allocation of the jobs to suitable factories. Therefore, two decisions have to be taken; job assignment to factories and job scheduling at each factory. Obviously, both problems are intimately related and cannot be solved sequentially if high performance is desired.

In this paper we study the distributed generalization of the PFSP, i.e., the distributed permutation flowshop scheduling problem or DPFSP in short. More formally, the DPFSP can be defined as follows: a set N of n jobs have to be processed on a set F of f factories, each one of them contains the same set M of m machines, just like in the regular PFSP. All factories are capable of processing all jobs. Once a job $j, j \in N$ is assigned to a factory $f, f \in F$ it cannot be transferred to another factory as it must be completed at the assigned plant. The processing time of job j on machine i at factory f is denoted as $P_{j,i,f}$. We are going to assume that processing times do not change from factory to factory and therefore, $P_{j,i,1} = P_{j,i,2} = \dots = P_{j,i,f} = P_{j,i}, \forall i \in M$. We study the minimization of the maximum makespan among all factories and denote the DPFSP under this criterion as $DF/prmu/C_{max}$. The total number of possible solutions for the DPSP is $\binom{n-1}{f-1}n!$, much more than the regular $n!$ solutions of the PFSP. Note that for reasons of space, we do not demonstrate this result here. Furthermore, for makespan minimization, no factory should be left empty with no jobs assigned. The demonstration for this is also omitted.

Rubén Ruiz

Grupo de Sistemas de Optimización Aplicada. Instituto Tecnológico de Informática (ITI).
Ciudad Politécnica de la Innovación, Edificio 8G. Acceso B. Universidad Politécnica de
Valencia. Camino de Vera s/n, 46022 Valencia, Spain.
E-mail: rruiz@eio.upv.es

Bahman Naderi

Department of Industrial Engineering, Amirkabir University of Technology, 424 Hafez
Avenue, Tehran, Iran.
E-mail: bahman.naderi@aut.ac.ir

We present a simple Variable Neighborhood Descent (VND) method (Mladenovic and Hansen, 1997) for this problem, together with some heuristic methods. The choice of the VND is motivated by the simplicity of this approach, which is a desirable characteristic of solution methods. Our intention is to provide some initial results for this interesting problem generalization of the PFSP.

2 Variable Neighborhood Descent method

In Variable Neighborhood Descent (VND), several different neighborhoods are explored in order typically from the smallest and fastest to evaluate, to the slower and bigger one. The process iterates over each neighborhood while improvements are found, doing local search until local optima at each neighborhood. Only strictly better solutions are accepted after each neighborhood search. Our proposed VND incorporates one local search to improve the job sequence at each factory, and another one to explore job assignments to factories. In the following we further detail the proposed VND.

2.1 Initial solution

Starting VND from a random solution quickly proved to be a poor decision since the method rapidly converges to a local optimum of poor quality. Recall that VND is a very simple form of local search. Furthermore, effective heuristic initialization of metaheuristic algorithms is a very common approach in the scheduling literature. As a result, we initialize the proposed VND from an adaptation of the well known NEH method from Nawaz et al. (1983). Basically, we follow all steps of the NEH, the only difference is that each job in the insertion list is inserted in all positions of all factories and is finally placed in the position resulting in the minimum makespan among all factories.

2.1 Solution representation and local search

Solution representation is a key issue for every metaheuristic method. In the regular PFSP, a job permutation is the most widely used representation. Recall that the DPFSP adds the job factory assignment dimension and therefore, such representation is not enough. We propose a representation composed by a set of F lists, one per factory. Each list contains a partial permutation of jobs, indicating the order in which they have to be processed at that factory. Notice that this representation is complete and indirect. It is complete because all possible solutions for the DPFSP can be represented and it is indirect since we need to decode the solution in order to calculate the makespan.

For the proposed VND we have two different neighborhoods. The first one is the typical insertion neighborhood where all assigned jobs to one factory are extracted, one by one, and reinserted in all possible sequence positions at that factory. Moves are accepted as long as improvements in the makespan of a given factory are obtained. This process is repeated for all factories. Therefore, the first local search works over the job sequencing decision of the DPFSP. The second local search deals with the factory assignment dimension. All jobs are extracted from their currently assigned factories and are inserted in all possible positions of all other factories. The acceptance criterion for this second neighborhood is more complicated. We propose two techniques: a) accept movements if the global makespan is decreased or b) accept movements if a “net gain” is achieved, i.e., we permit one of the two factories (origin or destination) to increase its makespan value provided that the other factory decreases its makespan and that the net effect is negative (increased makespan is less than decreased makespan).

Both proposed neighborhoods are very fast to evaluate as the traditional PFSP accelerations for makespan calculations can also be used in the DPFSP.

3 Computational results

We evaluate both proposed VND methods (each one applying one of the two proposed movement acceptance rules for the second local search, i.e., VNDa and VNDb) over a comprehensive benchmark. The benchmark contains 720 instances being the maximum size 500 jobs, 7 factories and 20 machines per factory. We include in the comparison straightforward extensions of existing methods from the literature (extended for the DPFSP). We measure the relative percentage deviation of each method from the best solution known for each instance. All tests have been carried out in a Core 2 Duo computer running at 2.0 GHz. The average results (proven to be statistically different) are given in the following table:

	SPT	LPT	Johnson	CDS	PALMER	NEH	VNDa	VNDb
Deviation	16.79	29.93	10.55	8.20	8.85	1.03	0.10	0.43
Time (sec)	<0.001	<0.001	<0.001	<0.001	<0.001	0.026	0.147	0.096

Table 1 – Average results of the proposed NEH and VND methods, compared with standard heuristics from the PFSP literature.

As we can see, all three proposed methods, NEH, VNDa and VNDb provide very good results. The best proposed method is VNDa, which gives the best solutions on average. The CPU time employed is extremely short as it is only 0.147 seconds for the slowest proposed method.

4 Conclusions

To the best of our knowledge, this paper represents the first attempt at solving an interesting generalization of the permutation flowshop problem: the distributed permutation flowshop problem. Several identical factories are available to process the jobs and an additional decision of job factory assignment has to be carried out. We have presented an extension of the well known NEH heuristic and two simple Variable Neighborhood Descent search methods based on two simple neighborhood structures. The presented methods are very fast and provide much better results than typical heuristic approaches. Of course, much more work needs to be done in this interesting problem. Exact approaches are needed in order to gauge the maximum solvable instance size. More advanced metaheuristics could improve the results even further. Also, the assumption that all factories are identical is not realistic and varying processing times when changing from factory to factory should be investigated. Lastly, minimizing the maximum makespan across factories is a good start but more work needs to be done in order to determine other plausible objectives like minimizing the sum of makespans or even the standard deviation of the makespans across factories.

References

- Mladenovic, N. and Hansen, P. (1997). "Variable neighborhood search", *Computers & Operations Research*, 24 (11):1097-1100
- Kahn, K. B., Castellion, G. A. and Griffin, A. (2004). *The PDMA Handbook of New Product Development*, Second edition, Wiley, New York.
- Moon, C., Kim, J. and Hur, S. (2002). "Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain", *Computers & Industrial Engineering*, 43 (1-2):331-349.
- Nawaz, M., Ensore, E. E. Jr., and Ham, I. (1983). "A Heuristic Algorithm for the m-Machine, n-Job Flow-Shop Sequencing Problem", *OMEGA, The international Journal of Management Science*, 11(1):91-95.
- Wang, B. (1997). *Integrated product, process and enterprise design*, Chapman & Hall, London.

Decision Support Systems for Tankering within the Airline Industry

Barry McCollum • Graham Kendall • Paul McMullan • Joe Mc Goldrick

1 Introduction

In recent years, the aircraft industry has attracted the interest of optimisation and scheduling researchers [1]. As with most real world problems there are many issues that have to be addressed in order to capture the complexities of the real world, and this is true of many of the problems faced by the airline industry. In this work we are particularly interested in the optimisation of fuel usage by airlines. This problem is known as tankering.

Aircraft fuel represents a significant cost to the industry and, for some airlines, it is their largest operating cost. The fuel costs for British Airways, for example, for the financial year ending in 2007 was £1,931M [2]. This represented a 22.1% increase over the previous year, which had already seen a 44.7% increase from the preceding year. In comparison, BA's employee costs were only slightly higher (£2,227M), with the increase over the previous year being only 0.8%. This suggests that although airlines have control over employee costs, the market dictates their fuel costs. Based on a 2007 US report on the aviation industry, it is estimated that the factors surrounding tankering are responsible for an annual wastage of 5% of aircraft fuel in the United States alone. The annual US usage of fuel was 21 billion gallons (80 billion litres) in 2006. Currently the cost of the fuel is £2.75 a gallon. Assuming current fuel usage is on a similar scale to 2006, within the US, tanking represents wastage of £2.89 Billion. Worldwide, it is estimated this could be as much as ten times this figure [3].

Operational data such as flight routes, when and where to re-fuel, how much fuel to take on at any given service point, landing charges and potential passenger load etc. are all routinely collected by airline companies. However, current software used within the airline industry is unable to offer effective optimisation of fuel usage. In addition, no effective decision support system is able to offer advice on route planning, advice on speed/altitude and where to refuel.

Takering encompasses how airlines seek to minimise their fuel costs by attempting to optimise various operational factors. For example, flight routes, when and where to re-fuel,

Barry McCollum
Queen's University Belfast
E-mail: B.McCollum@qub.ac.uk

Graham Kendall
The University of Nottingham
E-mail: gxk@cs.nott.ac.uk

Paul McMullan
Queen's University Belfast
E-mail: p.mcmullan@qub.ac.uk

Joe Mc Goldrick
Aircraft Management Technologies
E-mail: Joe.McGoldrick@flightman.com

how much fuel to take on at any given service point, landing charges and potential passenger load. It is considered essential by the industry that any optimisation approach should also provide robust solutions allowing for unforeseen circumstances such as aircraft availability, breakdown etc.

Currently, decisions relating to the amount of fuel carried and where and when to refuel are based on only a small proportion of the information available i.e. the captain can choose to refuel at various scheduled stops, which may not lead to the most efficient fuel usage. When considered across the entire industry, this suboptimal approach leads to massive fuel waste through burn off and financial waste through differential fuel pricing. In addition, it suggests that there is a need to optimise routes in the future based on minimising fuel costs, while considering all other objectives e.g. maximising passenger load etc.

Our industrial collaborator, Aircraft Management Technologies (AMT), is an award winning provider of software technology to automate and streamline operations within the aviation industry. Their flagship product, FlightmanTM, offers a family of products comprising innovative software technology which synchronises an extensive data model between the aircraft and various airline ground systems. It also provides forms and workflow tailored to various users to enable dramatic process improvement in the cockpit, cabin and on the ramp, thus reducing costs and enhancing efficiency in aircraft operations and maintenance.

Our research plans, working in collaboration with AMT, aims to address the tankering problem, by defining it as an optimisation problem where the objective is to minimise fuel costs. In its simplest form, the problem is one of accurately balancing fuel load against distance to the next location. However, the real world problem provides more complex scenarios. One important factor, which currently exists, is the captain's overestimate of the amount of fuel required due to the existence of a *comfort factor*. When an aircraft is taking fuel on board, in addition to price, factors which must also be considered are location, fuel availability, loading, charges and planned future routing (which includes the countries which are flown over as this has implications for charging).

The present state of our research is to fully understand the problem, by consultation with our industrial partner. At the conference we will further define the problem, and present any modeling and solution methodologies we have considered. We will also plan to undertake a data collection phase so that we can evaluate our proposed methodologies on real world data, rather than generated data which might not accurately reflect the real world.

References

1. C. Barnhart, P. Belobaba, and A.R. Odoni, Application of operations research in air transport industry, *Transportation Science* 37 (2003) pages 368-391.
2. British Airways 2006/2207 Annual Report and Accounts. (2007) Available from http://media.corporate-ir.net/media_files/irol/69/69499/Annual_Report_and_Accounts.pdf (last accessed 14th June 2007)
3. Stolzer A.J. (2002) Fuel Consumption Modelling of a Transport Category Aircraft using Flight Operations Quality Assurance: A Literature Review, *Journal Air Transportation*, 7(1): 93-102

Response time analysis in distributed real-time systems

An improvement based on best-case finalization time analysis

R.J. Bril · L. Cucu-Grosjean · J. Goossens

1 Introduction

Existing end-to-end response time analysis in distributed real-time systems [2], where the finalization of one task on a processor activates another task on another processor, is pessimistic. By “pessimistic” we mean that not all systems deemed to be unschedulable by the analysis are in fact unschedulable. This pessimism has two causes: (i) the existing analysis is based on best-case response times rather than best-case finalization times and (ii) those best-case response times are based on analysis for (worst-case) deadlines at most equal to periods minus (absolute) activation jitter [1]. In this paper, we present analytical means to determine best-case finalization times of independent real-time tasks with deadlines larger than periods minus activation jitter under uniprocessor fixed-priority preemptive scheduling (FPPS) and arbitrary phasing, allowing an improvement of the existing analysis. We will illustrate the improvement by means of an example.

We assume a single processor and a set $\mathcal{T}$ of n periodically released, independent tasks $\tau_1, \tau_2, \dots, \tau_n$ with unique, fixed priorities. At any moment in time, the processor executes the highest priority task that has work pending, i.e. tasks are scheduled using FPPS.

Each task τ_i generates an infinite sequence of jobs τ_{ik} with $k \in \mathbb{Z}$. The inter-activation times of τ_i are characterized by a (fixed) *period* $T_i \in \mathbb{R}^+$ and an (*absolute*) *activation jitter* $AJ_i \in \mathbb{R}^+ \cup \{0\}$, where $AJ_i < T_i$. Moreover, τ_i is characterized by a (fixed) *computation time* $C_i \in \mathbb{R}^+$, a *phasing* $\phi_i \in \mathbb{R}$, a (*relative*) *worst-case deadline* $WD_i \in \mathbb{R}^+$, and a (*relative*) *best-case deadline* $BD_i \in \mathbb{R}^+ \cup \{0\}$, where $BD_i \leq WD_i$. The set of phasings ϕ_i is termed the phasing ϕ of the task set $\mathcal{T}$. We assume that we have no control over the phasing ϕ , so any arbitrary phasing may occur. The deadlines BD_i and WD_i are relative to the activations.

Reinder J. Bril

Technische Universiteit Eindhoven (TU/e), Mathematics and Computer Science Department, Den Dolech 2,
5600 AZ Eindhoven, The Netherlands.

E-mail: R.J.Bril@tue.nl

Liliana Cucu-Grosjean

INRIA Nancy-Grand Est, TRIO team, 615 rue du Jardin Botanique, Villers les Nancy, 54600, France.

E-mail: Liliana.Cucu@loria.fr

Joël Goossens

Université Libre de Bruxelles (ULB), Computer Science Department, Boulevard du Triomphe - C.P.212, 1050
Brussels, Belgium.

E-mail: Joel.Goossens@ulb.ac.be

Note that the activations of τ_i do not necessarily take place strictly periodically with period T_i , but somewhere in an interval of length AJ_i that is repeated with period T_i . The activation times a_{ik} of τ_i satisfy $\sup_{k,\ell} (a_{ik}(\varphi_i) - a_{i\ell}(\varphi_i) - (k - \ell)T_i) \leq AJ_i$, where φ_i denotes the start of the interval in which job zero is activated, i.e. $\varphi_i + kT_i \leq a_{ik} \leq \varphi_i + kT_i + AJ_i$. A task with activation jitter equal to zero is termed a *strictly periodic* task.

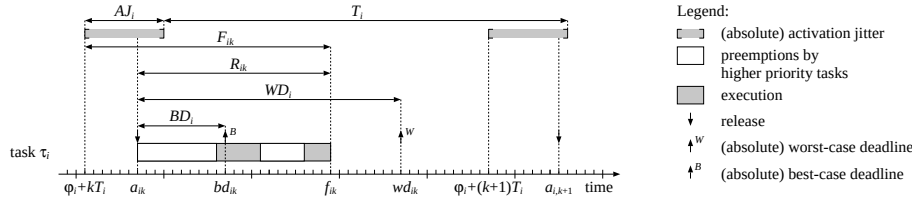


Fig. 1 Basic model for a periodic task τ_i with (absolute) activation jitter AJ_i .

The (*relative*) *finalization time* F_{ik} of job τ_{ik} is defined relative to the start of the interval in which τ_{ik} is activated, i.e. $F_{ik} = f_{ik} - (\varphi_i + kT_i)$. The *active interval* of job τ_{ik} is defined as the time span between the activation time of that job and its finalization time, i.e. $[a_{ik}, f_{ik}]$. The *response time* R_{ik} of job τ_{ik} is defined as the length of its active interval, i.e. $R_{ik} = f_{ik} - a_{ik}$. Figure 1 illustrates the above basic notions for an example job of a periodic task τ_i . Whereas $F_{ik} = R_{ik}$ holds for a strictly periodic task τ_i , the following relation holds in general

$$F_{ik} \geq R_{ik}. \quad (1)$$

For notational convenience, we assume that the tasks are given in order of decreasing priority, i.e. task τ_1 has highest priority and task τ_n has lowest priority.

2 Existing results

The *worst-case response time* WR_i and the *best-case response time* BR_i of a task τ_i are the largest and the smallest (relative) response time of any of its jobs, respectively, i.e. $WR_i \stackrel{\text{def}}{=} \sup_{\varphi,k} R_{ik}(\varphi)$ and $BR_i \stackrel{\text{def}}{=} \inf_{\varphi,k} R_{ik}(\varphi)$. For worst-case deadlines at most equal to periods minus activation jitter, i.e. $WD_i \leq T_i - AJ_i$, BR_i is given by the *largest* $x \in \mathbb{R}^+$ that satisfies

$$x = C_i + \sum_{j < i} \left(\left\lceil \frac{x - AJ_j}{T_j} \right\rceil - 1 \right)^+ C_j. \quad (2)$$

Here, the notation w^+ stands for $\max(w, 0)$, which is used to indicate that the number of preemptions of tasks with a higher priority than τ_i can not become negative. To calculate BR_i , we can use an iterative procedure based on recurrence relationships, starting with an upper bound, e.g. WR_i .

The *worst-case finalization time* WF_i and the *best-case finalization time* BF_i of a task τ_i are the largest and the smallest (relative) finalization time of any of its jobs, respectively, i.e. $WF_i \stackrel{\text{def}}{=} \sup_{\varphi,k} F_{ik}(\varphi)$ and $BF_i \stackrel{\text{def}}{=} \inf_{\varphi,k} F_{ik}(\varphi)$. The *worst-case (absolute) finalization jitter* FJ_i of task τ_i is the largest difference between the finalization times of any two of its jobs, i.e.

$$FJ_i \stackrel{\text{def}}{=} \sup_{\varphi,k,\ell} (F_{ik}(\varphi) - F_{i\ell}(\varphi)). \quad (3)$$

Finalization jitter analysis presented in [2] is based on $FJ_i \leq WF_i - BR_i$, where BR_i is determined using Equation (2).

3 Contributions

From Equation (3) we derive $FJ_i \leq WF_i - BF_i$. The finalization jitter analysis presented in [2] is therefore pessimistic for two reasons; firstly $BF_i \geq BR_i$ for $AJ_i > 0$; see Equation (1) and secondly BR_i as determined by Equation (2) is pessimistic for worst-case deadlines larger than periods minus activation jitter [1]. We will illustrate this by means of an example and subsequently present a conjecture for best-case finalization time analysis.

Table 1 presents the characteristics of our example task set $\mathcal{T}_1$ consisting of three tasks, and Figure 2 shows a time-line for $\mathcal{T}_1$ with $BF_3 = 3$ and $BR_3 = 2.4$ of task τ_3 , hence $BF_3 > BR_3$. Using Equation (2) yields a value $BR_3 = 2$, which is pessimistic, i.e. too small. A conjecture for exact best-case response time analysis of tasks with arbitrary deadlines that are scheduled using FPPS has been presented in [4].

task	T	C	AJ	BF	BR
τ_1	4	2	0	2	2
τ_2	5	1	0	1	1
τ_3	7	2	0.6	3	2.4

Table 1 Task characteristics of $\mathcal{T}_1$ and values for best-case finalization times and response times.

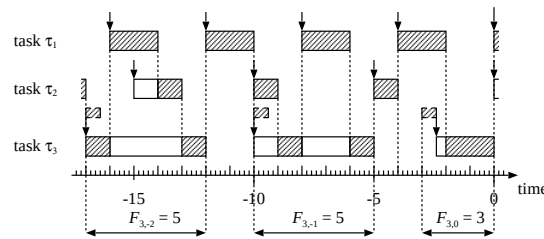


Fig. 2 Time-line for $\mathcal{T}_1$ with a best-case finalization time $BF_3 = 3$ and a best-case response time $BR_3 = 2.4$ for job $\tau_{3,0}$ of task τ_3 .

Conjecture 1 The best-case finalization time BF_i of task τ_i with $T_i - AJ_i < WD_i$ is given by

$$BF_i = \max_{0 \leq k < w\ell_i} (BR'_i((k+1)C_i) - kT_i). \quad (4)$$

where $w\ell_i$ is the worst-case number of jobs of τ_i in a level- i active period¹, and $BR'_i((k+1)C_i)$ is the best-case response time of a task τ'_i with a computation time $C'_i = (k+1)C_i$, a period equal to its worst-case deadline, i.e. $T'_i = WD'_i$, a worst-case deadline WD'_i given by

$$WD'_i = WD_i + \begin{cases} 0 & k = 0 \\ kT_i - AJ_i & \text{otherwise} \end{cases}, \quad (5)$$

¹ A level- i active period is the longest interval in which the sum of pending loads is higher than 0 for tasks with a priority equal to or higher than the priority of task τ_i ; see [3]. The length of the longest level- i active period is finite for all $1 \leq i \leq n$ when either (i) the utilization factor $U^T = \sum_{1 \leq j \leq n} \frac{C_j}{T_j}$ is smaller than 1 or when (ii) U^T is equal to 1, the activation jitter of all tasks of $\mathcal{T}$ are equal to zero, and the least common multiple of the periods of all tasks of $\mathcal{T}$ exists.

and a best-case deadline BD_i equal to its computation time, i.e. $BD_i = (k + 1)C_i$. $\square$

Based on Conjecture 1, we find $w\ell_3 = 3$ and $BF_3 = \max(2, 9 - 7, 17 - 14) = 3$.

Note that for $w\ell_i = 1$, (4) becomes equal to the solution of (2). Hence, the conjecture therefore applies for tasks with arbitrary deadlines.

References

1. O. Redell and M. Sandfridson, *Exact best-case response time analysis of fixed priority scheduled tasks*, Proc. 14th Euromicro Conference on Real-Time Systems, 165-172 (2002)
2. R. Henia, R. Racu and R. Ernst, *Improved Output Jitter Calculation for Compositional Performance Analysis of Distributed Systems*, Proc. IEEE Int. Parallel and Distributed Processing Symp., 1-8 (2007)
3. R.J. Bril, J.J. Lukkien and W.F.J. Verhaegh, *Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption*, Accepted for Real-Time Systems Journal (2009), <http://www.springerlink.com/content/f05r404j63424h27/fulltext.pdf> (online).
4. R.J. Bril, L. Cucu-Grosjean and J. Goossens, *Exact best-case response time analysis of real-time tasks under fixed-priority pre-emptive scheduling for arbitrary deadlines* Accepted for Work-in-Progress session of the 21st Euromicro Conference on Real-Time Systems, (2009).

Schedulability Analysis for non Necessarily Harmonic Real-Time Systems with Precedence and Strict Periodicity Constraints using the Exact Number of Preemptions and no Idle Time

Patrick Meumeu Yomsî · Yves Sorel

1 Introduction

Scheduling theory as it applies to hard real-time environments with precedence and strict periodicity constraints — environments where the failure to satisfy any constraint may have disastrous consequences [1], [2] — seems currently to be enjoying a renaissance. The most widely studied problems in this field concern periodic non-preemptive tasks for systems where data are collected through captors and the control is performed through actuators [3], [4]. The tasks corresponding to captors and actuators must satisfy a strict periodicity constraint. For reasons of consistency and predictability, we assume in this paper that all tasks are subject to this constraint. The domains considered here include automobiles, avionics, mobile robotics, telecommunications, etc. Although preemptive scheduling algorithms are able to successfully schedule some systems that cannot be scheduled by any non-preemptive scheduling algorithm, the cost of preemption may not be negligible. Therefore, when preemption is allowed, its exact cost has to be explicitly considered in the schedulability conditions in order to avoid wasting resources and provide safety in terms of guaranteeing the right behavior of the system at run-time. In this paper, we address the scheduling problem of hard real-time systems composed of **dependent, strictly periodic**, preemptive tasks in the monoprocessor case. The strictly periodic constraint implies that, for such a system, any task starts its execution at the beginning of its period whereas the dependence constraint implies that any task cannot start its execution before the end of another task preceding it. We assume here that no jitter is allowed at the beginning of each task. To clearly distinguish between the specification level and its associated model, we shall use the term *operation* rather than the commonly used “*task*” [5] which is too closely related to the implementation level.

For systems with the above-mentioned constraints, in [6] we proved that some of them can be eliminated because they are definitely not schedulable, then we solved the problem

INRIA Paris-Rocquencourt,
Domaine de Voluceau BP 105, 78153 Le Chesnay Cedex - France
E-mail: patrick.meumeu@inria.fr

INRIA Paris-Rocquencourt,
Domaine de Voluceau BP 105, 78153 Le Chesnay Cedex - France
E-mail: yves.sorel@inria.fr

for systems with harmonic periods¹ in [7]. Here, we first generalize these results to the case of systems with periods that are not necessarily harmonic. Then, we provide a necessary and sufficient schedulability condition which takes into account the exact number of preemptions for a system with such constraints when no idle time is allowed. That means the processor always executes an operation if there is one to execute. Indeed, even though the cost α of one preemption — the context switching time including the storage as well as the restoration of the context that the processor needs when a preemption occurs — is easy to know for a given processor, it remains a challenging problem to count the exact number of preemptions of each instance for a given operation [8], [9], [7]. As in [10], we consider only predictable processors without cache or complex internal architecture. We consider a set of n strictly periodic preemptive operations $\tau_i, 1 \leq i \leq n$ with precedence constraints. Each operation τ_i is an infinite sequence of instances² $\tau_i^k, k \in \mathbb{N}^+$, and is characterized by a Worst Case Execution Time (WCET) C_i , not including any approximation of the cost of preemption, as is usually the case in the classical real-time scheduling theory [11], [12], [13], [14], and a period T_i . In each instance τ_i^k , operation τ_i has to execute for C_i time units. The context switching time corresponding to the activation as well as the termination of an operation is constant, and thus is included in the WCET. Regarding the constraints, we have the following information.

The **precedence** constraint is given by a partial order on the execution of the operations. An operation τ_i preceding an operation τ_j is denoted by $\tau_i \prec \tau_j$ which means that $s_i^k \leq s_j^k, \forall k \geq 1$ thanks to the result given in [4], s_i^k denotes the start time of τ_i^k . In that paper it was proved that given two operations $\tau_i = (C_i, T_i)$ and $\tau_j = (C_j, T_j)$: $\tau_i \prec \tau_j \implies T_i \leq T_j$. Consequently, the operations must be scheduled in an increasing order of their periods corresponding to classical fixed priorities, using Rate Monotonic (RM), where the shorter the period the higher the priority [7], [4]. We re-index operations in such a way that $\tau_1 \prec \tau_2 \prec \dots \prec \tau_n$, that is to say τ_1 precedes τ_2 , τ_2 precedes τ_3 and so on. In the context of this paper we shall use the term “level” rather than priority, level 1 which corresponds to operation τ_1 being the highest, and level n which corresponds to operation τ_n being the lowest.

The **strict periodicity** constraint means that the start times s_i^k and s_i^{k+1} of two consecutive instances corresponding to operation τ_i are **exactly** separated by its period: $s_i^{k+1} - s_i^k = T_i, \forall k \geq 1$. The instance started at time $s_i^1 + kT_i$ has $s_i^1 + (k+1)T_i$ as its deadline.

For such a system of operations with precedence and strict periodicity constraints, we propose a method to compute on the one hand the exact number of preemptions, and on the other hand the schedule of the system when no idle time is allowed, i.e. the processor will always execute an operation as soon as it is possible to do so. Although idle time may help the system to be schedulable, when no idle time is allowed it is easier to find the start times of all the instances of an operation according to the precedence relation.

For the sake of readability and without any loss of generality, from now on, although it is not entirely realistic, we will consider the cost of one preemption for the processor to be $\alpha = 1$ time unit. It is worth noticing that the analysis performed here would work even if the preemption cost were not a constant.

The remainder of the paper is structured as follows: section 2 describes the model and gives the notations used throughout this paper. Section 3 provides the definitions we need to take into account the exact number of preemptions in the schedulability analysis presented in section 4. That section explains in detail, on the one hand, our scheduling algorithm

¹ A sequence $(a_i)_{1 \leq i \leq n}$ is harmonic if and only if there exists $q_i \in \mathbb{N}$ such that $a_{i+1} = q_i a_i$. Notice that we may have $q_{i+1} \neq q_i \quad \forall i \in \{1, \dots, n\}$.

² Throughout the paper all subscripts refer to operations whereas all superscripts refer to instances.

which counts the exact number of preemptions and, on the other hand, derives the new schedulability condition. We conclude and propose future work in section 5.

2 Model

Throughout the paper, we assume that all timing characteristics are non-negative integers, i.e. they are multiples of some elementary time interval (for example the “CPU tick”, the smallest indivisible CPU time unit). We denote by $\tau_i = (C_i, T_i)$: an operation, T_i : Period of τ_i , C_i : WCET of τ_i without any preemption approximation, $C_i \leq T_i$, α : Temporal cost of one preemption for a given processor, τ_i^k : The k^{th} instance of τ_i , $N_p(\tau_i^k)$: Exact number of preemptions of τ_i in τ_i^k , $C_i^k = C_i + N_p(\tau_i^k) \cdot \alpha$: Preempted Execution Time (PET) of τ_i including its exact preemption cost in τ_i^k , s_i^1 : Start time of the first instance of τ_i , $s_i^k = s_i^1 + (k-1)T_i$: Start time of τ_i^k , R_i^k : Response time of τ_i^k .

A *valid schedule* S for the system taking into account the exact number of preemptions will be yielded by the set of start times of the first instance for all operations: $S = \{(s_1^1, s_2^1, \dots, s_n^1)\}$. Since all the operations except the one with the shortest period w.r.t. the precedence relations may be preempted, the execution time of an operation may vary from one instance to another due to the number of preemptions. Therefore, the *preempted execution time* (PET) [10] which corresponds to the WCET augmented with the exact cost due to preemptions for each instance of an operation may also vary from one instance to another. Consequently, the PET denoted C_i^k for instance τ_i^k depends on the instance and on the number of preemptions occurring in that instance. Its computation will be detailed below.

Because we intend to take into account the exact number of preemptions, and because all operations may be preempted, except the first one, i.e. the one with the shortest period, all instances of all operations must be considered since the number of preemptions may be different from one instance to another. We give a schedulability condition for each operation individually according to operations with shorter periods. For each operation, our scheduling algorithm first provides the start time of the first instance, then computes the exact number of preemptions per instance. This individual operation analysis leads, at the end, to a schedulability condition for all operations.

It has been shown in [4], [7] that systems with precedence and strict periodicity constraints repeat identically after a time called the *hyperperiod* which corresponds to the Least Common Multiple (LCM) of the periods of all the operations.

3 Definitions

All the definitions and terminologies used in this section are directly inspired by [10] and are applied here to the case of a model with precedence and strict periodicity constraints.

From the point of view of any operation τ_i , we define the *hyperperiod at level i* , H_i , which is given by $H_i = LCM\{T_j\}_{\tau_j \in sp(\tau_i)}$, where $sp(\tau_i)$ is the set of operations with a period shorter than or equal to that of operation τ_i . The set $sp(\tau_i)$ may include τ_i . It is obvious that H_i time units after the first start time s_i^1 of operation τ_i , the start time of the next instance is exactly the same as that of s_i^1 w.r.t. the start time of the first instances of operations preceding τ_i . This characteristic derives from both the precedence and the strict periodicity constraints. Without any loss of generality we assume that the first operation τ_1 starts its execution at time $t = 0$. Since at each level i the schedule of τ_i repeats indefinitely, it is sufficient to perform the scheduling analysis in the interval $[s_i^1, s_i^1 + H_i]$ for τ_i and $[0, s_n^1 + H_n]$ for the

whole set of operations. Therefore, τ_i starts σ_i times in each hyperperiod at level i starting from 0, with $\sigma_i = \frac{H_i}{T_i} = \frac{LCM\{T_j\}_{\tau_j \in sp(\tau_i)}}{T_i}$.

Because operation τ_i may only be preempted by the set of operations with a period shorter than τ_i denoted $sp(\tau_i)$, then there are exactly σ_i different PETs for operation τ_i . In other words, from the point of view of any operation τ_i , we can define the function π , inflating C_i , as $\pi: \mathbb{N}^+ \times \mathbb{N}^+ \longrightarrow \mathbb{N}^{+\sigma_i} \times \mathbb{N}^+$, where $\pi(C_i, T_i) = \pi(\tau_i) = ((C_i^1, C_i^2, \dots, C_i^{\sigma_i}), T_i)$, which maps the WCET C_i of operation τ_i into its respective PET C_i^k in each instance τ_i^k when τ_i is schedulable.

Because of the precedence constraints among operations and because we proceed the schedule from the operation with the shortest period towards the operation with the longest period. At each priority level the goal is to fill available time units in the previous schedule thus far obtained, with slices of the WCET of the current operation taking into account the exact number of preemptions, and hence we obtain the next current schedule. Consequently, we represent the previous schedule of every instance τ_i^k of the current operation $\tau_i = (C_i, T_i)$ by an ordered set of T_i time units where some are already executed because of the execution of operations with shorter periods relatively to $\prec$, and the others are still available for the execution of operation τ_i in that instance. We call this ordered set which describes the state of each instance τ_i^k the $\mathcal{M}_i^k$ T_i -mesoid. We denote a time unit already executed by an “e” and a time unit still available by an “a”. The switch from an a to an e represents a preemption if the WCET of the current operation is strictly greater than the cardinal of the sub-set corresponding to the first sequence of a . Depending on the remaining execution time while filling available time units, this situation may occur again leading therefore to several preemptions which themselves may result in causing others. The cardinal of a sub-set corresponding to a sequence of consecutive time units already executed is called a *consumption*. It will be denoted by its value inside brackets. We enumerate the sequence of available time units according to natural numbers. This enumeration is done from the end of the first sequence of time units already executed in that instance. Each of these natural numbers corresponds to the number of available time units since the end of the first consumption. They represent all the possible PETs of the operation under consideration in the corresponding instance. Each of these natural numbers a_i is called an *availability*. For example, the 13-mesoid $\{e, e, e, a, a, a, e, e, a, a, e, a, a\}$ will be represented by $\{(3), 1, 2, 3, (2), 4, 5, (1), 6, 7\}$, $(3), (2), (1)$ are consumptions and $1, 2, 3, 4, 5, 6, 7$ are availabilities. More details on the definition of a T_i -mesoid are given in [10].

From the point of view of the current operation $\tau_i = (C_i, T_i)$, there are as many T_i -mesoids as instances in the hyperperiod H_i at level i . Therefore, there are σ_i T_i -mesoids in H_i which will form a sequence of T_i -mesoids. We call $\mathcal{L}_i^b = \{\mathcal{M}_i^{b,1}, \mathcal{M}_i^{b,2}, \dots, \mathcal{M}_i^{b,\sigma_i}\}$ the sequence of σ_i T_i -mesoids **before** τ_i is scheduled in the current schedule. The process used to build the sequence $\mathcal{L}_i^b$ of operation τ_i will be detailed later. We define for each T_i -mesoid $\mathcal{M}_i^{b,k}$ the corresponding *universe* X_i^k to be the ordered set, compatible with that of the corresponding mesoid, which consists of all the availabilities of $\mathcal{M}_i^{b,k}$. That is to say, all the possible values that C_i^k can take in $\mathcal{M}_i^{b,k}$. Recall that C_i^k denotes the PET of τ_i in τ_i^k , the k^{th} instance of τ_i . Operation τ_i will be said to be *potentially schedulable* if and only if

$$\begin{cases} C_i \in X_i^k & \forall k \in \{1, \dots, \sigma_i\} \\ \mathcal{M}_i^{b,k} \text{ starts with an available time unit for each } k \in \{1, \dots, \sigma_i\} \end{cases} \quad (1)$$

The first σ_i equations of (1) verify that C_i belongs to each universe at level i . Then, the next σ_i equations verify that every $\mathcal{M}_i^{b,k}$ starts with an availability as no idle time is allowed.

These verifications are necessary for the strict periodicity constraints to be satisfied. As a matter of fact, if a T_i -mesoid starts with a consumption it is not possible to fill the previous schedule with slices of the WCET of the current operation τ_i taking into account the cost of preemption as it belongs to a lower level than those already scheduled w.r.t $\prec$. Therefore its start time is postponed to the end of the consumption in the previous schedule, and thus does not satisfy the strict periodicity constraint of τ_i . In this case the system is not schedulable. Notice that when this situation arises the three non schedulability conditions given in [6] hold. Since $C_i \in \{1, 2, \dots, T_i\}$, $\forall 1 \leq i \leq n$, let us define the following binary relation on each instance.

$\mathcal{R}$: “availability a_{i_1} leads to the same number of preemptions as availability a_{i_2} ”,
 $a_{i_1}, a_{i_2} \in \{1, 2, \dots, T_i\}$

$\mathcal{R}$ is clearly an equivalence relation on $\{1, 2, \dots, T_i\}$ (reflexive, symmetric, transitive). Now, since $X_i^k \subseteq \{1, 2, \dots, T_i\}$, $\forall 1 \leq k \leq \sigma_i$, thus $\mathcal{R}$ is also an equivalence relation on X_i^k and each X_i^k together with $\mathcal{R}$ is a *setoid*³. From now on, we consider only the restriction of $\mathcal{R}$ on X_i^k , $k = 1, \dots, \sigma_i$ because X_i^k represents all the available time units in instance τ_i^k .

Each T_i -mesoid consists of a sequence of time units already executed, i.e. consumptions, due to the schedule of operations with shorter periods, followed or preceded by a sequence of times units still available, i.e. availabilities. Since each switch from an available time unit to an already executed time unit possibly corresponds to a preemption, then according to the value of C_i several preemptions may occur. Among the possible values that C_i can take, those which will lead to the same number of preemptions will be said to be equivalent w.r.t. to $\mathcal{R}$, and thus will belong to the same *equivalence class*. Therefore, the *equivalence classes* of each universe correspond to the subsets of availabilities determined by two consecutive consumptions in the associated mesoid. The start time of the first instance s_i^1 of operation τ_i occurs at least after the end time of that of operation τ_{i-1} in order to satisfy the strict periodicity constraint. Moreover, s_i^1 occurs as soon as possible since no idle time is allowed. The latter statement implies that operation τ_i starts Δ_{i-1} time units after the start time s_{i-1}^1 of the first instance of operation τ_{i-1} . The computation of Δ_{i-1} will be detailed later on. Already, it is worth noticing that Δ_{i-1} is longer than or equal to the response time of τ_{i-1} in its first instance because when the last piece of the PET of τ_{i-1} fits exactly a sequence of consecutive availabilities, then the start time of the first instance of τ_i is postponed. Hence, we can derive the first start time of any potentially schedulable operation τ_i as the **sum** of the start time s_{i-1}^1 of the first instance of operation τ_{i-1} and Δ_{i-1} : $s_i^1 = s_{i-1}^1 + \Delta_{i-1}$.

When equation (1) holds for a given operation τ_i , we call $\mathcal{L}_i^a = \{\mathcal{M}_i^{a,1}, \mathcal{M}_i^{a,2}, \dots, \mathcal{M}_i^{a,\sigma_i}\}$ the sequence of σ_i T_i -mesoids of operation τ_i **after** τ_i is scheduled. $\mathcal{L}_i^a$ is a function of $\mathcal{L}_i^b$ which itself is a function of $\mathcal{L}_{i-1}^a$, both detailed as follows. Thanks to everything we have presented up to now, we can assume without any loss of generality that the start time of the first instance of the operation with the shortest period, here τ_1 , starts its execution at time $t = 0$, i.e. $s_1^1 = 0$.

Let f be the function such that $\mathcal{L}_i^b = f(\mathcal{L}_{i-1}^a)$ which transforms the sequence $\mathcal{L}_{i-1}^a$ of σ_{i-1} T_{i-1} -mesoids after operation τ_{i-1} has been scheduled at level $i - 1$ into the sequence $\mathcal{L}_i^b$ of σ_i T_i -mesoids before operation τ_i is scheduled at level i .

As mentioned above, a mesoid consists only of time units already executed denoted by “e” and time units still available denoted by “a”. Moreover, the cardinal of a mesoid is equal to the period of the operation under consideration whatever the level is. As such, the

³ A setoid is a set equipped with an equivalence relation.

function f transforms a time unit already executed (resp. still available) in the sequence $\mathcal{L}_{i-1}^a$ into a time unit already executed (resp. still available) in the sequence $\mathcal{L}_i^b$ by following an index ψ which enumerates according to natural numbers, the time units (already executed or still available) in the sequence $\mathcal{L}_{i-1}^a$ of operation τ_{i-1} after τ_{i-1} is scheduled. ψ starts from the first available time unit of the first mesoid $\mathcal{M}_{i-1}^{a,1}$ towards the last time unit of the last mesoid $\mathcal{M}_{i-1}^{a,\sigma_{i-1}}$, and then circles around to the beginning of the first mesoid $\mathcal{M}_{i-1}^{a,1}$ again, until we get the σ_i T_i -mesoids of $\mathcal{L}_i^b$. During this process each time $\psi = T_i$, a T_i -mesoid is obtained for the sequence $\mathcal{L}_i^b$ and then the next T_i -mesoid is obtained by starting to count again from the next time unit to the current one. Indeed, the previous schedule at level i (the schedule obtained at level $i-1$) consists of H_{i-1} time units whereas the schedule of the current operation τ_i is computed upon H_i time units after the start time of its first instance s_i^1 . That amounts to extending the previous schedule from H_{i-1} to H_i time units by identically repeating the previous schedule as often as necessary to obtain H_i time units.

Due to the precedence and strict periodicity constraints, notice that ψ in contrast to index ζ used in [10] which started from the first time unit, starts from the first available unit of the first T_{i-1} -mesoid as no idle time is allowed. The value of Δ_{i-1} is therefore the consumption before the first available time unit in the sequence $\mathcal{L}_{i-1}^a$ of operation τ_{i-1} .

Since τ_1 is the operation with the shortest period, $sp(\tau_1) = \{\tau_1\}$ and thus $\sigma_1 = \frac{H_1}{T_1} = 1$.

Moreover, because τ_1 is never preempted, we have $\mathcal{L}_1^b = \{\mathcal{M}_1^{b,1}\} = \{\{1, 2, \dots, T_1\}\}$ and $\mathcal{L}_1^a = \{\mathcal{M}_1^{a,1}\} = \{\{(C_1), 1, 2, \dots, T_1 - C_1\}\}$.

Let g be the function such that $\mathcal{L}_i^a = g(\mathcal{L}_i^b)$ which transforms the sequence $\mathcal{L}_i^b$ of σ_i T_i -mesoids before operation τ_i has been scheduled at level i into the sequence $\mathcal{L}_i^a$ of σ_i T_i -mesoids after operation τ_i has been scheduled at level i .

For each T_i -mesoid $\mathcal{M}_i^{b,k}$ of $\mathcal{L}_i^b$ we compute the PET C_i^k that we add to all the consumptions appearing in that T_i -mesoid before the availability corresponding to that PET. This yields the response time R_i^k of operation τ_i in instance τ_i^k . The PET is computed by using a fixed-point algorithm which is detailed in the next section. Now, for each T_i -mesoid of $\mathcal{L}_i^b$, function g transforms a time unit already executed in the sequence $\mathcal{L}_i^b$ into a time unit already executed in the sequence $\mathcal{L}_i^a$, and transforms a time unit still available into either a time unit still available or a time unit already executed w.r.t. the following condition. We use an index which enumerates using numerals the time units from the first to the last one in each T_i -mesoid $\mathcal{M}_i^{b,k}$ of $\mathcal{L}_i^b$. If the current value of the index is less than or equal to R_i^k , then function g transforms the time unit still available into a time unit already executed due to the execution of instance τ_i^k , otherwise g transforms it into a time unit still available. Indeed, function g fills available time units in the current schedule with slices of the PETs in each T_i -mesoid, leading to the previous schedule for the next operation at level $i+1$ w.r.t. $\prec$.

4 The proposed approach

In this section, we outline our approach for a system with an arbitrary number of operations. This approach leads to a new schedulability condition for hard real-time systems with precedence and strict periodicity constraints using the exact number of preemptions and no idle time allowed. This condition is new in the sense that it takes into account the exact number of preemptions in the schedulability analysis for such systems rather than using an approximation in the WCETs.

Since the schedule proceeds from the operation with the shortest period corresponding to the highest level, to the one with the longest period corresponding to the lowest level, then for every potentially schedulable operation, we determine its schedule thanks to those with shorter periods. At each priority level i , the basic idea consists in filling availabilities in each mesoid of the sequence $\mathcal{L}_i^b$, before operation τ_i is scheduled, with slices (cardinal of equivalence classes) of its inflated WCET while taking into account the cost of the exact number of preemptions. At each preemption occurrence, α time units add to the remaining execution time of the instance of the operation under consideration. This situation may occur again w.r.t. the remaining execution time, leading therefore to several preemptions which themselves may cause others. This is why it is crucial to calculate the exact number of preemptions. Finally, we obtain for each mesoid the PET, and then the corresponding response time. Determining the worst case among these response times allow us to conclude on the schedulability of operation τ_i w.r.t. $\prec$. When τ_i is schedulable, we build the sequence $\mathcal{L}_i^a$, after τ_i is scheduled, in order to check the schedulability of the next operation, and so on, otherwise the system is not schedulable.

4.1 Scheduling algorithm

We assume that the first $i - 1$ operations with $2 \leq i \leq n$ have already been scheduled, i.e. the sequence $\mathcal{L}_{i-1}^a$ of operation τ_{i-1} is known, and that we are about to schedule operation τ_i , which is potentially schedulable, i.e. equation 1 holds.

Sequence $\mathcal{L}_i^b = f(\mathcal{L}_{i-1}^a)$ of operation τ_i is built thanks to index ψ on sequence $\mathcal{L}_{i-1}^a$ of operation τ_{i-1} without forgetting to start at the first available time unit rather than the first time unit as in [10]. Again this is due to the constraints on the system and the fact that no idle time is allowed: the start time of the first instance of operation τ_i is at $s_i^1 = s_{i-1}^1 + \Delta_{i-1}$. The sequence $\mathcal{L}_i^b$ consists of σ_i T_i -mesoids $\mathcal{M}_i^{b,k}$ with $k = 1, \dots, \sigma_i$ since operation τ_i may only be preempted by operations belonging to $sp(\tau_i)$. We can therefore determine the universes $X_i^k \forall k \in \{1, \dots, \sigma_i\}$ when the sequence $\mathcal{L}_{i-1}^a$ is known. The response time R_i^k of operation τ_i in its k^{th} instance, i.e. in the k^{th} T_i -mesoid will be obtained by summing C_i^k with all consumptions prior to C_i^k in the corresponding mesoid. The worst-case response time R_i of operation τ_i will be given by $R_i = \max_{\{1 \leq k \leq \sigma_i\}} (R_i^k)$. This equation leads us to say that operation τ_i is schedulable if and only if $R_i \leq T_i$. Again, $\mathcal{L}_i^a = g(\mathcal{L}_i^b)$ will be deduced from sequence $\mathcal{L}_i^b$ like $\mathcal{L}_1^a = g(\mathcal{L}_1^b)$ in the previous section. For the sake of clarity, whenever there are two consecutive consumptions in the same *mesoid*, this amounts to considering only one consumption which is the sum of the previous consumptions.

Below, we present our scheduling algorithm which, for a given operation, on the one hand first determines the start time of its first instance relatively to $\prec$, then counts the exact number of preemptions in each of its instances, and on the other hand provides its PET in each of its instances in order to take into account the exact number of preemptions in the schedulability condition. It has the following twelve steps. Since the operation with the shortest period, namely operation τ_1 , is never preempted, the loop starts from the index of the operation with the second shortest period, namely operation τ_2 .

- 1: **for** $i = 2$ to n **do**
- 2: Compute the number σ_i of times that operation $\tau_i = (C_i, T_i)$ has started in the hyper-period at level i .

- 3: Determine the start time of the first instance of operation τ_i : $s_i^1 = s_{i-1}^1 + \Delta_{i-1}$ where Δ_{i-1} is the consumption before the first available time unit in the sequence $\mathcal{L}_{i-1}^a$ of operation τ_{i-1} .
- 4: Build the sequence $\mathcal{L}_i^b = f(\mathcal{L}_{i-1}^a)$ of T_i -mesoids of operation τ_i before it is scheduled. This construction consists of σ_i T_i -mesoids $\mathcal{M}_i^{b,k}$ with $k = 1, \dots, \sigma_i$, and is based on a modulo T_i arithmetic using index ψ on the sequence $\mathcal{L}_{i-1}^a$ without forgetting to start at the first available time unit rather than the first time unit as in [10].
- 5: For each T_i -mesoid $\mathcal{M}_i^{b,k}$ resulting from the previous step, build the corresponding universe X_i^k which consists of the ordered set of all availabilities of $\mathcal{M}_i^{b,k}$. Notice that this set corresponds to the set of all possible values that the PET C_i^k of operation τ_i can take in $\mathcal{M}_i^{b,k}$.
- 6: Build all the equivalence classes for each universe X_i^k . An equivalence class of X_i^k is composed of the subset of availabilities determined by two consecutive consumptions in the associated mesoid $\mathcal{M}_i^{b,k}$. $m \in \mathbb{N}$ in expression $[m]^k$ denotes the subset of X_i^k composed of the availabilities which are preempted m times.
- 7: Compute both the exact number of preemptions and the PET C_i^k of operation τ_i in each universe X_i^k , $1 \leq k \leq \sigma_i$, resulting from the previous step thanks to the algorithm inlined in this step. Since τ_i is potentially schedulable, i.e. its WCET C_i belongs to one and only one equivalence class $[\theta_1]^k$ in each universe X_i^k , we must verify that it is actually schedulable given that some preemptions may occur. The recursive inflation of the execution time of the operation, due to preemptions, starts from the value of the WCET. Indeed, the current inflated WCET is obtained by adding the previous inflated WCET and the cost of preemptions incurred by this latter WCET. This explains the following fixed-point algorithm.

$$\begin{cases} \theta_0 = 0 \\ C_i^{k,0} = C_i \in [\theta_1]^k \\ C_i^{k,m} = C_i^{k,m-1} + (\theta_m - \theta_{m-1}) \cdot \alpha \in [\theta_m]^k \quad \forall m \geq 1 \end{cases}$$

This computation stops as soon as either the PET is reached, i.e. two consecutive values of $C_i^{k,j}$, $j \geq 1$ are equal, or there exists $\mu_2 \geq 1$ such that $C_i^{k,\mu_2} > \text{card}(X_i^k)$. In

the latter case, operation τ_i is not schedulable. In the first case, $C_i^k = C_i + \sum_{j=1}^l (\theta_j -$

$$\theta_{j-1}) \cdot \alpha = C_i + N_p(\tau_i^k) \cdot \alpha.$$

- 8: Deduce the image $\tau_i' = \pi(\tau_i) = ((C_i^1, C_i^2, \dots, C_i^{\sigma_i}), T_i)$ of operation τ_i resulting from the previous step.
- 9: Determine the response time R_i^k , $1 \leq k \leq \sigma_i$ of operation τ_i in its k^{th} instance, i.e. in the k^{th} T_i -mesoid. This is obtained by summing C_i^k with all the consumptions prior to C_i^k in the corresponding mesoid. Deduce the worst-case response time R_i of operation τ_i : $R_i = \max_{\{1 \leq k \leq \sigma_i\}} (R_i^k)$. Operation τ_i is schedulable if and only if $R_i \leq T_i$.
- 10: If $R_i \leq T_i$ then build the sequence $\mathcal{L}_i^a = g(\mathcal{L}_i^b)$, increment i , and go back to step 2 as long as there remain potentially schedulable operations in the system.
- 11: If $R_i > T_i$, then the system $\{\tau_i = (C_i, T_i)\}_{1 \leq i \leq n}$ is not schedulable.
- 12: **end for**

Thanks to the above algorithm, a system $\{\tau_i = (C_i, T_i)\}_{1 \leq i \leq n}$, with precedence and strict periodicity constraints where no idle time is allowed and which takes into account the exact number of preemptions, is schedulable if and only if $R_i \leq T_i$ for all $i \in \{1, 2, \dots, n\}$.

5 Conclusion and future work

We are interested in hard real-time systems with precedence and strict periodicity constraints where it is mandatory to satisfy these constraints. We are also interested in preemption which offers great advantages when seeking schedules. Since classical approaches are based on an approximation of the cost of the preemption in WCETs, possibly leading to an incorrect real-time execution, we proposed an approach that takes its exact cost into account. We proposed a scheduling algorithm which counts the exact number of preemptions for a given system and thus gives a stronger schedulability condition than those in the literature.

Currently, we are adding the latency constraints to our model and we are planning to study the same problem when jitter is allowed on the periods of operations and then, the complexity of our approach. Afterwards, because idle time may increase the possible schedules, we also plan to allow idle time, even though this would increase the complexity of the scheduling algorithm.

References

1. Joseph Y.-T. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 1980.
2. Ray Obenza and Geoff. Mendal. Guaranteeing real time performance using rma. *The Embedded Systems Conference, San Jose, CA*, 1998.
3. K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems*, 1995.
4. L. Cucu and Y. Sorel. Schedulability condition for systems with precedence and periodicity constraints without preemption. In *Proceedings of 11th Real-Time Systems Conference, RTS'03*, Paris, March 2003.
5. J.H.M. Korst, E.H.L. Aarts, and J.K. Lenstra. Scheduling periodic tasks. *INFORMS Journal on Computing* 8, 1996.
6. P. Meumeu Yomsi and Y. Sorel. Non-schedulability conditions for off-line scheduling of real-time systems subject to precedence and strict periodicity constraints. In *Proceedings of 11th IEEE International Conference on Emerging technologies and Factory Automation, ETFA'06*, WIP, Prague, Czech Republic, September 2006.
7. P. Meumeu Yomsi and Y. Sorel. Schedulability analysis with exact number of preemptions and no idle time for real-time systems with precedence and strict periodicity constraints. In *Proceedings of 15th International Conference on Real-Time and Network Systems, RTNS'07*, Nancy, France, March 2007.
8. Burns A., Tindell K., and Wellings A. Effective analysis for engineering real-time fixed priority schedulers. *IEEE Trans. Software Eng.*, 1995.
9. J. Echagüe, I. Ripoll, and A. Crespo. Hard real-time preemptively scheduling with high context switch cost. In *Proceedings of the 7th Euromicro Workshop on Real-Time Systems*, 1995.
10. P. Meumeu Yomsi and Y. Sorel. Extending rate monotonic analysis with exact cost of preemptions for hard real-time systems. In *Proceedings of 19th Euromicro Conference on Real-Time Systems, ECRTS'07*, Pisa, Italy, July 2007.
11. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.
12. J. Goossens. Scheduling of offset free systems. *The Journal of Real-Time Systems*, 2003.
13. J. Goossens and Devilliers R. The non-optimality of the monotonic priority assignments for hard real-time offset free systems. *Real-Time Systems*, 1997.
14. M. Grenier, J. Goossens, and N. Navet. Near-optimal fixed priority preemptive scheduling of offset free systems. *14th International Conference on Real-time and Network Systems*, Poitiers, 2006.

Speeding up the optimal off-line algorithm for solving the k -server problem

Tomislav Rudec · Alfonzo Baumgartner ·
Robert Manger

1 Introduction

In the k -server problem [6] one has to decide how k mobile servers should serve a sequence of requests. Each server occupies a location in a fixed, possibly infinite, metric space. Repeatedly, a new request appears at some location x . To serve the request, a corresponding algorithm must move a server to x unless it already has a server at that location. Whenever the algorithm moves a server from a location x to a location y , it incurs a cost equal to the distance between x and y . The goal of good serving is not only to serve requests, but also to minimize the total distance moved by all servers.

It is usually required that the solution to the k -server problem is produced in *on-line* fashion [4], so that each request is served before the next request arrives. However, for various reasons, it is also useful to consider *off-line* solutions. Contrary to any on-line procedure, whose serving is never quite satisfactory due to lack of information on future requests, an off-line procedure knows the whole input in advance and can therefore provide optimal serving.

2 Aim of the paper

In this paper we are concerned with the *optimal* off-line algorithm for solving the k -server problem - let us call it OPT. It is well known [2] that OPT can be implemented quite efficiently by network flow techniques. The aim of this paper is to propose certain modifications to the original network flow implementation of OPT. Our modifications improve the original version in terms of speed. The aim of the paper is also to evaluate experimentally the obtained improvements.

Tomislav Rudec

Faculty of Electrical Engineering, University of Osijek, Trpimirova 2b, 31000 Osijek, Croatia
E-mail: Tomislav.Rudec@etfos.hr

Alfonzo Baumgartner

Faculty of Electrical Engineering, University of Osijek, Trpimirova 2b, 31000 Osijek, Croatia
E-mail: Alfonzo.Baumgartner@etfos.hr

Robert Manger

Department of Mathematics, University of Zagreb, Bijenička 30, 10000 Zagreb, Croatia
E-mail: Robert.Manger@math.hr

3 Original implementation

As it has been described in [2], OPT finds the optimal way of serving a given sequence of n requests by k servers with given initial positions. The solution is obtained by computing the *minimal-cost maximal flow* on a suitably constructed network with $2n + k + 2$ nodes. Actual computation of the optimal flow is accomplished by the *flow augmentation* method [1], which reduces to exactly k steps. Each of those steps reduces to a single-source shortest path problem in a so-called displacement network consisting again of $2n + k + 2$ nodes. With some preprocessing [3], any shortest path problem can be solved by Dijkstra's procedure [5].

4 Modified implementation

Our modified implementation of OPT is obtained from the original implementation by modifying each of its steps separately. All modifications are based on special properties of the involved networks. Here are some details.

- In each step arcs or nodes are identified which cannot influence the remaining part of the algorithm. Such arcs or nodes are removed from the network.
- In the first step it is observed that the sought shortest path is almost completely determined in advance, only one arc has yet to be chosen. Consequently, the shortest path is constructed directly, without any general path-finding procedure, by considering k possibilities and finding the minimum among certain k values.
- In the second step it is observed that the displacement network is acyclic. Thus the shortest path is found by simple scanning of nodes in topological order, which is several times faster than with Dijkstra.
- In the third, fourth, or any of the remaining steps the shortest path is found by a slightly customized version of Dijkstra. Moreover, if it happens that the shortest path has length 0, then the whole algorithm is immediately stopped since the current solution cannot change anymore.

Our modifications produce substantial improvement of the first two steps of OPT. The improvement of the remaining steps is not so dramatic. Still, thanks to constant removal of arcs and nodes, the network in each step becomes simpler, thus enabling faster execution.

5 Experimental results

Our experiments have been designed to measure the overall effect of all modifications applied to OPT. We have tested both versions of OPT on the same set of k -server problem instances, with the request sequence length n ranging from 1000 to 3000, and the number of servers k ranging from 2 to 20. Some results of our experiments are presented in Table 1, where each entry corresponds to one particular problem instance with particular values of n and k . According to the table, the modified version of OPT turns out to be considerably faster than the original version, achieving the speedup between 5 and 13. The experiments have also shown that for a fixed n the speedup becomes larger as k becomes smaller.

Table 1 Speedup of the modified OPT vs. the original OPT

	$n = 1000$	$n = 1500$	$n = 2000$	$n = 2500$	$n = 3000$
$k = 2$	13.625	9.957	10.615	9.504	9.112
$k = 3$	10.097	9.026	8.865	7.744	7.563
$k = 5$	8.565	7.712	7.179	6.795	6.861
$k = 10$	6.453	6.728	6.689	6.140	6.373
$k = 20$	6.466	6.521	6.411	5.955	6.028

6 Conclusion

The optimal off-line algorithm for the k server problem is important, because it provides benchmarks for on-line computation, and because it serves as a building block for certain complex on-line algorithms that rely on solving auxiliary off-line subproblems. In this paper we have described how the conventional network-flow implementation of the optimal off-line algorithm can be improved in terms of speed. According to our experimental results, the improved version of the algorithm is indeed considerably faster than the original version, specially if the number of servers k is relatively small.

References

1. M.S. Bazaraa, J.J. Jarvis, H.D. Sherali, Linear Programming and Network Flows, Third edition, Wiley-Interscience, New York NY (2004).
2. M. Chrobak, H. Karloff, T.H. Payne, S. Vishwanathan, New results on server problems, SIAM Journal on Discrete Mathematics, Vol 4, 172-181 (1991).
3. J. Edmonds, R.M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, Journal of the ACM, Vol 19, 248-264 (1972).
4. S. Irani, A.R. Karlin, Online computation, in: Approximation Algorithms for NP-Hard Problems, (D. Hochbaum, Ed.), 521-564, PWS Publishing Company, Boston MA (1997).
5. D. Jungnickel, Graphs, Networks and Algorithms, Second edition, Springer, Berlin (2005).
6. M. Manasse, L.A. McGeoch, D. Sleator, Competitive algorithms for server problems, Journal of Algorithms, Vol 11, 208-230 (1990).

Author Index

Abd-Aziz	823
Abdul-Hamid	709
Abdullah	424, 727
Aggoune	469
Al-Betar	157
Alidaee	765
Almeder	105, 706
Alsheddy	130
Atkin	12
Ayob	399
Babić	311
Baker	330
Balasubramanian	357
Barbosa-Póvoa	802
Barták	507, 518, 768
Baumgartner	852
Beliën	724
Belouadah	202
Benabid	254
Berghman	752
Berlińska	321
Bilgin	683
Billaut	85
Bini	77
Błażewicz	748
Borgulya	827
Bouzouia	227
Bożejko	702
Brauner	679
Briand	227
Bril	839
Burke	12, 409, 424, 435, 727, 737, 784, 788, 790, 798
Byde	720
Čangalović	311
Ceschia	594
Chakhlevitch	782
Chang	531
Chase	341
Chen M-C	531
Chen S-H	531
Chen Y-M	531
Chrétienne	291
Ciavotta	808
Côté	831
Creemers	756
Cucu-Grosjean	839

Curtois	790
Darties	378
Dasaklis	570
Dauzère-Pérès	192, 581
De Causmaecker	683, 693
De Reyck	756
Demeulemeester	724
Denton	696
Detienne	581
Dožić	311
Dror	28
Drozdowski	321
Duša	518
Elvikis	645
Finke	679
Fleszar	202
Fowler	192, 357, 696
Gafarov	671
Gao	765
Gather	811
Gendreau	831
Giroudeau	378
Glass	782
Gomes	802
Gonzalez	720
Goossens D.	715
Goossens J.	839
Greenwood	12
Guedes	115
Guitouni	277
Gul	696
Gunawan	145
Günther	741
Hamacher	645
Hanen	254
Hanzálek	239
Hartl	706
Haspeslagh	693
Haugen	821
Hazir	291
He	435
Herbots	772
Herrmann	628
Hiranaka	655
Hurink	761
Huschka	696
Hyde	790, 798
Jain	720
Jat	180
Juraszek	266, 699
Kalić	311

Katzensteiner	706
Kedad-Sidhoum	291
Keha	357
Kendall	399, 709, 790, 823, 837
Kergosien	85
Khader	157
Khowala	357
Kok	761
König	378
Korošec	616
Kravchenko	458
Kvaratskhelia	68
Kwan	43, 778
Lambrecht	756
Lanquepin	821
Lau	145
Lauret	768
Lazarev	68, 671
Lehoux-Lebacque	679
Lenté	85
Leus	752, 756, 772
Lewis	718
Lin	764
Lingireddy	341
Løkketangen	821
Machowiak	748
Maenhout	690
Manger	852
Mason	192
Masri	277
Mati	469
McCollum	424, 727, 737, 778, 837
McGoldrick	837
McMullan	424, 737, 778, 837
Minella	808
Misir	784
Mokotoff	479
Mönch	105, 192
Müller	732
Murray	732
Naderi	834
Narin	744
Ng	606
Nishi	655
Nobibon	772
Novais	798
Ochoa	790
Oğuz	744, 748
Olstad	821
Ormsbee	341
Osman	202

Ourari	227
Özcan	784, 788, 798
Papa	616
Pappis	570
Parkes	424, 737
Paulus	761
Pawlak	775
Pesch	699
Petrovic	790
Pillay	447
Potvin	831
Qu	93, 409, 435
Rachaniotis	570
Rana	765
Ribeiro	115
Rose	192
Rucinski	775
Rudec	852
Rudová	732
Ruiz	808, 834
Sabar	399
Saffar	202
Sagir	709
Santos	720
Sato	216
Schaerf	594
Schutten	761
Schwiegelshohn	557
Serafini	59
Shabtay	301
Simonin	378
Skalický	507
Soghier	409
Sorel	843
Spieksma	715
Steiner	28, 301
Sterna	266, 699
Šůcha	239
Šumbera	768
T'kindt	645
Tanaka	216
Thompson	718
Tran	606
Trietsch	330
Tsang	130
Turabieh	727
Uchroński	702
Vakhania	366, 389
Vanden Berghe	683, 693
Vanhoucke	690
Vázquez-Rodriguez	790

Weeks	765
Werner	458
Wodecki	702
Xu	93
Yang	180
Yomsi	843
Yugma	581
Zerola	768
Zhang A.....	720
Zhang Q.....	531
Zimmermann	811