

WATT -- EURO Working Group On Automated TimeTabling



27-29 AUG 2013

MISTA 2013 PROCEEDINGS



ORBEL
The Belgian Operational Research Society

EVENTMAP

6th Multidisciplinary International Scheduling Conference: Theory & Applications, Gent, Belgium, 27-29 Aug 2013

MISTA 2013

Proceedings of the

6th Multidisciplinary International Conference on Scheduling: Theory and Applications

27th – 29th August 2013
Gent, Belgium

Edited by
Graham Kendall, University of Nottingham, UK
Greet Vanden Berghe, KAHO – KU Leuven, Belgium
Barry McCollum, Queens University Belfast, UK

MISTA 2013 Conference Program Committee

Abdullah, Salwani
Alvarez-Valdes, Ramon
Artigues, Christian
Ayob, Masri
Bai, Ruibin
Bartak, Roman
Blazewicz, Jacek
Burke, Edmund
Cai, Xiaoqiang
Carlier, Jacques
Chen, Zhi-Long
Chong, Siang Yew
Coffman, Ed
Cucu-Grosjean, Liliana
De Causmaecker, Patrick
Dell'Amico, Mauro
Dror, Moshe
Drozdowski, Maciej
Finke, Gerd
Froncek, Dalibor
Glover, Fred
Goossens, Dries
Gupta, Jatinder
Hao, Jin-Kao
Henz, Martin
Herrmann, Jeffrey W.
Hoogeveen, Han
Hyde, Matthew
Janiak, Adam
Jerry, Swan
Jouglet, Antoine
Kendall, Graham (conference chair)
Kingston, Jeffrey H.
Knust, Sigrid
Kubiak, Wieslaw
Kurz, Mary
Kwan, Raymond
Leung, Joseph
Levner, Eugene
Løkketangen, Arne
Mattfeld, Dirk
McCollum, Barry (conference chair)
McMullan, Paul
Moench, Lars
Mumford, Christine
Norman, Bryan
Osman, Ibrahim
Özcan, Ender
Pappis, Costas
Pesch, Erwin
Petrovic, Sanja
Pillay, Nelishia
Prins, Christian
Qu, Rong
Rayward-Smith, Vic J.
Ribeiro, Celso
Rudová, Hana
Sagir, Mujgan
Schaerf, Andrea
Thompson, Jonathan
Trautmann, Norbert
Trick, Michael
Trystram, Denis
Urrutia, Sebastián
Vanden Berghe, Greet (conference chair)
Vincent, T'Kindt
Wauters, Tony
Yugma, Claude
Zinder, Yakov

MISTA 2013 International Advisory Committee

- Graham Kendall (chair)
- Abdelhakim Artiba, Facultes Universitares Catholiques de Mons (CREGI - FUCAM), Belguim
- 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 a great 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 are grateful to the chairs of the special session on **Timetabling**, Barry McCollum and Hana Rudová, for taking the time to collect together an excellent set of papers.

We greatly appreciate the support that we have received from EventMap Ltd., ORBEL (The Belgian Operational Research Society) and WATT (EURO Working Group on Automated Timetabling).

We would also like to thank the Journal of Scheduling which supports the MISTA conference series by allowing us to guest edit a special issue of the journal which is associated with the conference. This certainly adds to the conference and the post-conference opportunities.

We are also grateful for the continued support we have received from the University of Nottingham (both UK and Malaysia campuses).

A dedicated team from the CODeS research group, KU Leuven supported the conference in several ways. We would like to thank Jan Christiaens, Joris Kinable, Mihail Mihaylov, Erik Van Achter and Sam Van Malderen who assisted with editorial and practical tasks both before and during the conference.

MISTA 2013 hosted a competition (see page 801). We wish to express special gratitude to Tony Wauters (chair), Pieter Smet, Wim Vancroonenburg and Jannes Verstichel for organizing the *MISTA 2013 challenge*, which attracted many researchers, from a variety of backgrounds.

We also owe a special thank you to Phyllis Moereels who has worked tirelessly for the social programme of MISTA 2013.

Finally, the conference owes a great deal to Debbie Pitchfork. She has worked tirelessly since performing similar tasks for MISTA 2009 and MISTA 2011 and if it were not for Debbie this conference would not have taken place. Thank you Debbie.

In Memory

In the past year, two of MISTA's greatest supporters have sadly passed away.

Peter Brucker (24th July 2013) has been on both the Program Committee and has served on the International Advisory since MISTA started in 2003. He has also published widely in MISTA, and its special issues. This is a list of his MISTA contributions:

1. Brucker, P *The Job-Shop Problem: Old and New Challenges*. In proceedings of the 3rd Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2007), 28 -31 August 2007, Paris, France, pages 15-22, 2007.
2. Brucker, P and Kampmeyer, T *Tabu Search Algorithms For Cyclic Machine Scheduling Problems with blocking*. In proceedings of the 2nd Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2005), 18 -21 July 2005, New York, USA, pages 107-108, 2005.
3. Brucker, P and Kampmeyer, T *Cyclic job shop scheduling problems with blocking*. Annals of Operations Research, 159 (1): 161-181, 2008.
4. Brucker, P; Qu, R; Burke, E and Post, G A *Decomposition, Construction And Post-processing Approach For Nurse Rostering*. In proceedings of the 2nd Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2005), 18 -21 July 2005, New York, USA, pages 397-406, 2005.

At MISTA 2005 (New York), he also presented a tutorial entitled *Shop Scheduling with Blocking*.

Peter was also scheduled to be a plenary speaker at MISTA 2013.

Of course, this only scratches the surface of his long and distinguished academic career and he will be greatly missed both professionally and personally.

Arne Løkketangen (10th June 2013) has served on the MISTA Program Committee since the conference series started in 2003. He has also contributed to MISTA 2009, with the following paper:

1. Løkketangen, A; Haugen, K; Lanquepin, G and Olstad, A *Using a set of fixed prices when solving Profit Maximization Capacitated Lot-Size Problems (PCLSP).* Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009), 10-12 Aug 2009, Dublin, Ireland, pages 821-822, 2009.

Arne had interests in meta-heuristic, combinatorial optimisation; with a particular focus on transportation. He was well respected in those communities, with a publication record that underpins that well deserved respect. The community will miss his scientific contributions, as well as his many other contributions.

On behalf of the MISTA community the Conference Chairs would like to express our gratitude to the many contributions that both Peter and Arne made to our community. Both will remain in our thoughts, especially during MISTA 2013, which is so close the passing of these two colleagues.

Table of Contents

Plenary Presentations

Della Croce F. *LP, or not LP: that is the question* 013

Potts C. *Planning and Scheduling for Railways: Review and Case Studies*.....014

Witteveen C. *Decomposition in scheduling*.....015

Papers

Sioud A., Gravel M. and Gagné C. *A new ant colony optimization for minimizing tardiness in the single machine scheduling problem with sequence-dependent setups times* 017

Messelis T., De Causmaecker P. and Vanden Berghe G. *Algorithm performance prediction for nurse rostering* 021

Quilliot A. and Chrétienne P. *Homogeneously non-idling schedules of unit-time jobs on identical machines* 039

Adasme P. and Lisser A., Wang C. and Soto I. *Scheduling in Wireless OFDMA-TDMA Networks using Variable Neighborhood Search MetaHeuristic* 055

Nishi T. and Sugiyama T. *Bilevel Decomposition Approach for Cyclic Staff Scheduling Problems* 065

Berlinska J. and Drozdowski M. *Mitigating Partitioning Skew in MapReduce Computations* ...

....080

Quilliot A. and Toussaint H. *RCPSP with Resource Transfers and Money Management* 091

Moura A. and Oliveira J. *Container Stowage and Short Sea Ship Routing Problem Optimization* 111

Briand C., Ngueveu S.U. and Sucha P. *Solving a cooperative project scheduling with controllable processing times, self-interested agents and equal profit sharing* 125

Agnetis A., Aloulou M.A. and Fu L-L. *Production and Inbound Batch Delivery Scheduling: Dominance and Cooperation* 133

Fernandes P., Pereira C.S. and Barbosa A. *Automatic timetabling in Higher Education Institutions: a real scenario and solution* 151

Tanaka S. *The single-machine scheduling problem with a finite capacity input buffer* 171

Deleplanque S. and Quilliot A. *Transfers in the on-demand transportation: the DARPT - Dial-a-Ride Problem with transfers allowed* 185

Jeauneau V. and Chrétienne P. *Mimizing the number of late jobs on parallel machines with alpha time windows* 206

Dostert M., Politz A. and Schmitz H. *Algorithms and Complexity of Student Sectioning for Existing Timetables* 218

Zeltzer L., Limère V. and Aghezzaf El-H *Balancing and Sequencing Mixed-Model Assembly Lines to Minimize Work Overload in Real World Workstations* 230

Benmansour R., Allaoui H., Artiba H. and Hanafi S. *Minimizing the weighted sum of maximum earliness- tardiness cost on a single machine under maintenance constraints* 240

Müller T. *Real-life Examination Timetabling* 248

Seddik Y., Gonzales C. and Kedad-Sidhoum S. *A polynomial time OPT-1 algorithm for a scheduling problem with two delivery dates and cumulative payoffs* 268

Gafarov E.R., Dolgui A., Lazarev A.A. and Werner F. *A Graphical Algorithm for Solving an Investment Optimization Problem* 290

Muehlenthaler M. and Wanka R. A *Decomposition of the Max-min Fair Curriculum-based Course Timetabling Problem: The Impact of Solving Subproblems to Optimality* 300

Bellio R., Ceschia S., Di Gaspero L., Schaerf A. and Urli T. *A simulated annealing approach to the curriculum-based course timetabling problem* 314

Geirsson E. and Runarsson T.P. *Bandit Algorithms using Monte Carlo Rollouts for Job Shop Scheduling* 318

Fonseca G.H.G., Santos H.G. and Toffolo T.A.M. *Late Acceptance-Hill Climbing Applied to the High School Timetabling Problem* 331

Fonseca G.H.G. and Santos H.G. *A New Integer Linear Programming Formulation to the Examination Timetabling Problem* 345

Pira C. and Artigues C. *Line search method for solving a non-preemptive strictly periodic scheduling problem* 356

Demir O. and Özdamar L. *A GIS-Based Emergency Response System for Disaster Relief Operations* 372

Laborie P. and Rogerie J. *Temporal Linear Relaxation in IBM ILOG CP Optimizer* 379

Valouxis C., Gogos C., Alefragis P., Goulas G., Voros N. and Housos E. *DAG Scheduling using Integer Programming in heterogeneous parallel execution environments* 392

Van Vreckem B., Borodin D., De Bruyn W. and Nowé A. *A Reinforcement Learning Approach to Solving Hybrid Flexible Flowline Scheduling Problems* 402

Martínez Y., Nowé A. and Bello R. *A Robust Multi-Agent Reinforcement Learning Approach for Scheduling Problems* 410

Komarudin, De Feyter T., Guerry M-A. and Vanden Berghe G. *Extended Roster Quality Staffing Problem: A medium term roster quality to support staffing decisions* 424

Scholz-Reiter B., Tan Y. and Hildebrandt T. *A computational study of shifting bottleneck procedures for job shop total weighted tardiness scheduling problems* 437

Rodrigues L.U., Santos H.G., Toffolo T.A.M. and Silva C.L.T.F. *Local search techniques applied to the Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Time* 458

Pineda A.A.S., Pecero J.E., Huacuja H.J.F., Barbosa J.J.G. and Bouvry P. *An Iterative Local Search Algorithm for Scheduling Precedence-Constrained Applications on Heterogeneous Machines* 472

Bucar D., Bessler S., Musliu N. and Groenbaek J. *Scheduling of electric vehicle charging operations* 486

Hamilton-Bryce R., McMullan P. and McCollum B. *Directing selection within an extended great deluge optimization algorithm* 499

Creemers S., Colen P. and Lambrecht M. *Evaluation of appointment scheduling rules* 509

Valero-Herrero M., Garcia-Sabater J.P. and Vidal-Carreras P.I. *In-depth analysis of the quality sequence to resolve CRSP* 515

Rahman S.A., Bargiela A., Burke E., McCollum B. and Özcan E. *An Investigation of Multistage Approaches to Examination Timetabling* 528

Weng F.C., bin Asmuni H. and McCollum B. *Tackling University Timetabling Problems Using Hybridization of Swarm Based Approach* 543

Sabar N.R., Ayob M. and Kendall G. *A Hybrid of Differential Evolution and Simulated Annealing Algorithms for the Capacitated Arc Routing Problems* 549

Klos T., Wilson M. and Witteveen C. *Flexibility Metrics for Time and Resource Constrained Scheduling Problems* 555

Abstracts

Nakamura S., Chen M. and Nakayama K. *Optimal Parallel Systems for Random Scheduling Times* 573

Kellerer H. and Strusevich V. *Fast Approximation Schemes for Boolean Programming and Scheduling Problems Related to Positive Convex Half-Product* 577

Van den Bergh J., Beliën J., Leus R. and Nobibon F.T. *A resource loading problem in the aircraft line maintenance industry* 580

Ermis G. and Oguz C. *Scheduling chains with constant timelags and variable release times on a single machine* 584

Bocquillon R. and Jouglet A. *Minimizing the dissemination length in the one-datum-unit data transfer problem* 587

Lapègue T., Bellenguez-Morineau O. and Prot D. *A large neighborhood search for the Shift Design Personnel Task Scheduling Problem with Equity* 590

Cheng C., He R., Yao F. and Yang Z. *Order acceptance and scheduling problem with time-dependent revenues* 593

Goossens D. and Spieksma F.C.R. *Scheduling round robin tournaments for amateur sport leagues* 596

Barth T.C., Böckmann F. and Schäfer A. *Baggage Scheduling at Airports: A framework for efficiently allocating resources to flight events* 599

Berghman L., Spieksma F.C.R. and T'Kindt V. *Solving a time-indexed formulation by preprocessing and cutting planes* 603

Curran D., Van der Krogt R., Little J. and Wilson N. *Compilation of Maintenance Schedules based on their Key Performance Indicators for Fast Iterative Interaction* 606

Amrouche K. and Boudhar M. *Reentrance and exact time lag on flow shop scheduling problem* 610

Kuhpfahl J. and Bierwirth C. *A GRASP approach for the Job Shop Scheduling Problem with minimizing the Total Weighted Tardiness* 613

Leoff J., Ackermann H. and Küfer K-H. *Time-Hierarchical Scheduling: A worst-case analysis of a rolling hierarchical approach for integrated production planning and scheduling in make-to-order environments* 617

Cerdeira J.O., Lopes I.C. and Costa e Silva E. *Optimal scheduling of aircrafts' engines repair process* 620

Wei W., Nobibon F.T. and Leus R. *Optimal scheduling policies for diagnosing k-out-of-n systems with imperfect tests* 624

Kristiansen S., Mason A. and Stidsen T.R. *Student Sectioning at High Schools in Denmark* 628

De Bruecker P., Demeulemeester E., Beliën J. and Van den Bergh J. *Algorithms for building optimal aircraft maintenance personnel rosters* 633

Gafarov E.R., Dolgui A. and Lazarev A.A. *Solution Algorithms for the Two-station Single Track Railway Scheduling Problem* 636

Sabar N.R. and Nazri M.Z.A *An Iterated Local Search Algorithm for the Technician Routing and Scheduling Problem* 641

Ceschia S. and Schaerf A. *The Dynamic Patient Admission Scheduling with Operating Room Constraints, Flexible Horizon, and Patient Delays* 646

Lancia G. and Serafini P. *Scheduling teams for aircraft ground maintenance* 649

Vancroonenburg W. and Wauters T. *Extending the late acceptance metaheuristic for multi-objective optimization* 652

Davari M., Demeulemeester E., Leus R. and Nobibon F.T. *Exact algorithms for single-machine scheduling with time windows and precedence constraints* 656

Vansteenwegen P. and Mateo M. *An efficient metaheuristic for the inventory orienteering problem and the single-vehicle cyclic inventory routing problem* 659

Smet P. and Vanden Berghe G. *A heuristic approach to an integrated personnel rostering and task assignment problem* 664

Blazewicz J., Bouvry P., Musial J., Pecero J.E. and Trystram D. *Dual Discounting Functions for Internet Shopping Optimization Problem* 667

Coolen K., Leus R. and Nobibon F.T. *Complexity analysis of the discrete sequential search problem with group activities* 670

Maenhout B. and Vanhoucke M. *An exact procedure for an integrated project staffing problem* 673

Verbeeck C., Aghezzaf E.-H and Vansteenwegen P. *A fast solution method for the Time-Dependent Orienteering Problem with time windows* 676

Deghdak K., T'Kindt V. and Bouquard J-L. *Evacuation Scheduling of a Town after a Natural Disaster: Time-indexed Formulations* 680

Ingels J. and Maenhout B. *The Impact of Problem Characteristics on the Robustness of a Personnel Schedule: An Empirical Investigation* 684

Dauzère-Pérès S. and Moench L. *New MILP Formulations for Minimizing the Number of Tardy Jobs on a Single Batch Machine* 687

Bykov Y. and Petrovic S. *An initial study of a novel Step Counting Hill Climbing heuristic applied to timetabling problems* 691

Hernández J.J.G., García M.J.G. and Hernández J.G.R. *Use Matrixes Of Weighing for the scheduling in a sports Fantasy* 694

Garraffa M., Salassa F., Vancroonenburg W., Vanden Berghe G. and Wauters T. *A computational study of a cutting stock problem with sequence dependent cut losses* 703

Guedes A.F.M., and Tereso A.P. *Scheduling with MS Project: an add-in* 707

Walcarius T., Houbraken M., Audenaert P., Pickavet M. and Demeester P. *Modeling Real-World Vehicle Routing Problems with Dependencies* 711

Walker S. and Zinder Y. *New Solvable Cases of the Maximum Lateness Problem with Preemptions* 714

Bitar A., Dauzère-Pérès S. and Yugma C. *An unrelated parallel machine scheduling problem with auxiliary resources and sequence dependent setup times* 718

Hanzalek Z. and Hanen C. *Grouping tasks to save energy in a cyclic wireless sensor network scheduling problem* 721

Mihaylov M., Wauters T. and Vanden Berghe G. *A 3-phase approach to geographically distributed exam timetabling* 725

Sahin G. and Saridarq F.D. *The Balanced Path Problem in Crew Scheduling Applications* 729

Han J., Sadykov R. and Vanderbeck F. *Parallel Machine Scheduling with Generalized Precedence Relations* 733

Beliën J. and Mercy A. *Building university course timetables with minimized resulting student flows: A case study at the Hogeschool-Universiteit Brussel* 737

Dauzère-Pérès S., Shen L. and Neufeld J.S. *Solving the Hybrid Job Shop Problem with Sequence Dependent Setups* 741

Vaclavik R., Sucha P. and Hanzalek Z. *Roster Evaluation Based on the Classifiers for the Nurse Rostering Problem* 744

Cortés R., Molina P. and Garcia-Sabater J.P. *On the effect of complexity and uncertainty in ELSP* 748

Lykourentzou I., Aggoune-Mtalaa W., Vergados D.J. and Naudet Y. *Guiding wiki crowds through resource scheduling* 753

Hazir O. and Kedad-Sidhoum S. *Batch Sizing and Just-In-Time Scheduling: Analysis of Some Special Cases* 757

Kochaniková B. and Rudová H. *Student Scheduling for Bachelor State Examinations* 762

Clark M.D. and Parkes A.J. *Heuristic Pruning Methods For The Robust Shortest Path Problem* 767

Elhag A. and Özcan E. *A Grouping Hyper-heuristic for Graph Colouring* 770

Lamas P. and Demeulemeester E. *The stochastic resource-constrained project scheduling problem under different confidence levels* 774

Muklason A., Parkes A.J., McCollum B. and Özcan E. *Initial Results on Fairness in Examination Timetabling* 777

Earl M. *Innovative teaching and the death of the utilisation survey* 781

Christiaens J., Wauters T., De Wispelaere A. and Vanden Berghe G. *A heuristic approach to the prisoner transportation problem* 786

Brauner N., Finke G., Shafransky Y. and Sledneu D. *Two extensions of Lawler's min-max cost algorithm* 790

Turky A.M., Abdullah S., McCollum B. and Sabar N.R. *An Evolutionary Hill Climbing Algorithm for Dynamic Optimisation Problems* 795

Aharon G. *Hierarchical Control Model for Several Stochastic Network Projects* 800

MISTA 2013 Challenge: Finalists

Schnell A. *A hybrid local search algorithm integrating an exact CP-SAT approach to solve the multi-mode resource-constrained multi-project scheduling problem* 802

Geiger M.J. *Iterated Variable Neighborhood Search for the resource constrained multi-mode multi-project scheduling problem* 807

Alonso-Pecina F., Pecero J.E. and Romero D. *A Three-Phases Based Algorithm for the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem* 812

Artigues C. and Hebrard E. *MIP Relaxation and Large Neighborhood Search for a Multi-Mode Resource-Constrained Multi-Project Scheduling Problem* 815

López-Ibáñez M., Mascia F., Marmion M-E. and Stützle T. *Automatic Design of a Hybrid Iterated Local Search for the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem* 820

Borba L.M., Benavides A.J., Zubaran T., Carniel G.C. and Ritt M. *A simple stochastic local search for multi-mode resource-constrained multi-project scheduling* 826

Bouly H., Dang D-C., Moukrim A. and Xu H. *Evolutionary algorithm and post-processing to minimize the total delay in multi-project scheduling* 831

Asta S., Karapetyan D., Kheiri A., Özcan E. and Parkes A.J. *Combining Monte-Carlo and Hyper-heuristic methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem* 836

Toffolo T.A.M., Santos H.G., Carvalho M. and Soares J. *An Integer Programming Approach for the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem* 840

Plenary Papers

Della Croce F. *LP, or not LP: that is the question* 849

Plenary Presentations

MISTA 2013

LP, or nor LP, that is the question

Federico Della Croce

Abstract

LP, or not LP: that is the question.
Whether 'tis nobler in the mind to suffer
The slings and arrows of linear programming,
Or to take arms against a sea of metaheuristics,
And, by opposing, end them? CPLEX, XPRESS; and more.

It is well known that in the 20th century, linear programming (LP) modeling, even though strongly applied to combinatorial optimization, was not too successful when directed to scheduling problems. The purpose of this talk is to show that the field of successful applications of LP (and MP - mathematical programming) modeling is still growing and includes also scheduling topics. We first focus on the single machine total completion time scheduling problem with different release times and present a matheuristic based on the related positional completion times LP formulation. Then, we show how LP can be used to tighten bounds for approximation results in scheduling theory. Similarly, we show how to enhance the complexity bounds of branch-and-reduce exact exponential algorithms by means of the so-called measure- and-conquer paradigm requiring always the solution of a specific MP model. Finally, we present a basic constructive LP-based approach to generate proper logic puzzles such as SUDOKU, FUTOSHIKI and TATAMI containing always a unique solution.

The full version of this paper is available later in the proceedings (see page 849).

MISTA 2013

Planning and Scheduling for Railways: Review and Case Studies

Chris Potts

MISTA 2013

Planning and Decomposition in Scheduling

Cees Witteveen

Abstract

Sometimes a scheduling problem has to be solved by distributed solvers. In this talk, I consider the use of scheduling decomposition methods that ensure a globally feasible outcome (a global schedule) by purely local computations, i.e., individual schedules composed by distributed solvers. Examples range from scheduling of reconnaissance missions and distributed violation detection in P2P systems, to railway scheduling problems. To study properties of these decomposition methods we model scheduling problems as constraint problems. Distributed scheduling problems then can be modeled by constraint systems where the set of variables is partitioned and each block of variables is controlled by a separate agent. Such an agent assigns values to the variables, and the aim is to provide distributed methods enabling a set of agents to come up with a global assignment (schedule) that satisfies all the constraints. We discuss some main approaches in such distributed scheduling, in particular the so-called temporal decoupling method and its generalizations. Then we discuss some general results regarding optimality and finding decompositions and we show that there exists a close connection between optimal decompositions and finding flexible schedules for certain classes of scheduling problems. Finally, we discuss some applications of decoupling in scheduling.

A paper related to this talk can be seen later in the proceedings (see page 555).

Papers

MISTA 2013

A new ant colony optimization for minimizing tardiness in the single machine scheduling problem with sequence-dependent setups times

Aymen Sioud · Marc Gravel · Caroline Gagné

1 Introduction

In industries such as metallurgy, petrochemicals, paper, aerospace, pharmaceuticals, electronics and ceramics, a machine bottleneck may affect some or all of the jobs in the production system. The handling of this machine is crucial for the enterprise because it is responsible for delays in customer's orders delivery [6]. Job scheduling on this machine is a way of approaching the problem and treating the whole production system. Also, the single machine model provides a basis for more complicated environment such as those seen in many real cases where researchers have remarked wide gaps between the theoretical problems and those met in practice [1,6]. In a survey of more than 250 industrial projects, Conner [2] has noted that 50% of these projects contain sequence-dependent setup times, and when these setup times are well applied, 92% of the order deadline could be met. Production of good schedules often relies on management of these setup times, and decision makers must therefore organize the job scheduling by trying to minimize downtime while respecting the different deadlines [6,1,2].

This present paper considers the minimization of total tardiness in the single machine scheduling problem with sequence dependent setup times (SMSDST). This problem, noted as $1|s_{ij}|\Sigma T_j$ in accordance with the notation of Graham *et al.* [7], is an NP-hard problem [8]. To solve this problem, we introduce a new ant colony optimization algorithm (ACO) which incorporates a transition rule featuring look ahead information and past information based on archive concept such as the multiobjective evolutionary computation. This new transition rule represents the main contribution of our study. The idea here is to adapt the transition rule to the treated problem.

Aymen Sioud, Marc Gravel
Département d'informatique et de mathématique, Université du Québec à Chicoutimi
555 Boulevard Université, Chicoutimi, Canada
E-mail: {asioud, mgravel}@uqac.ca

Caroline Gagné
Département Des Sciences économiques et gestion, Université du Québec à Chicoutimi
555 Boulevard Université, Chicoutimi, Canada
E-mail: c3gagne@uqac.ca

2 The ACO algorithm

Different approaches have been proposed by a number of researchers to solve the studied problem such as simulated annealing, GRASP, genetic algorithms and Tabu/VNS, among others. Only two ACO algorithms have been used to solve this problem [3,5]. However, they did not provide good results compared to the best approaches [9]. The main purpose of this work is to show that ACO algorithms can be efficient approaches for solving the $1|s_{ij}|\Sigma T_j$ problem when the different mechanisms of the algorithm are well designed to deal with the specificities of the problem, especially the transition rule.

After the pheromone initialization, where for all pairs of jobs *(i,j)* the trail intensity $\tau_{ij}(0)$ is initialized to a small positive quantity τ_0, the main loop in the proposed ACO algorithm has m ants construct a sequence of n jobs in five steps : *(i)* an initial job is set ; *(ii)* each ant builds a job sequence using the pseudo random proportional transition rule in Equations 1 and 2 ; *(iii)* a local pheromone update is performed ; *(iv)* a local improvement heuristic is applied ; and *(v)* a global pheromone update is applied. This main loop is executed for t_{max} cycles as shown in Figure 1.

```
/* STEP 0 : Pheromone Initialization */
for all job pair (i,j) do
    τ_ij(0) = τ_0
end for
/* Main Loop */
for t = 1 → t_max do
    /* STEP 1 : Set initial job */
    for k = 1 → m do
        Set the initial job for the ant k
    end for
    /* STEP 2 : Build a sequence */
    for i = 1 → n do
        for k = 1 → m do
            Choose the next job to insert using the Equations 1 and 2
            /* STEP 3 : Local pheromone update */
            for all chosen job pair (i,j) do
                τ_ij(t) = p_t * τ_ij(t) + (1 − p_t) * Δτ_ij(t) where Δτ_ij = τ_0
            end for
        end for
    end for
    /* STEP 4 : Local improvement */
    for k = 1 → m do
        Apply local improvement method or-opt heuristic
    end for
    /* STEP 5 : Global pheromone update */
    for all adjacent job pair (i,j) ∈ the best sequence Q* do
        τ_ij(t) = p_t * τ_ij(t) + (1 − p_t) * Δτ_ij(t) where Δτ_ij = 1/L*
    end for
end for
```

Fig. 1 THE ACO algorithm

From an existing partial job sequence each ant builds a sequence using the pseudo random proportional transition rule in Equations 1 and 2. In theses equations, the elements $\tau_{ij}(t)$ and s_{ij} represent the pheromone trail and the visibility, respectively. For the visibility we use the setup times between jobs. The element $SUCC_{ij}(A_t)$ represents the past information

which is introduced by a matrix built from an archive that stores the best solutions throughout the evolution process as in some cases in multi-objective evolutionary algorithms using the Pareto-optimal concept. Finally, the element H_{ij} represents the look ahead information which uses an heuristic that anticipates the choices in the transition rule. This heuristic is based on an upper bound of the total tardiness. For the local improvement heuristic we use the *or-opt* heuristic [10] adapted to the total tardiness.

$$j = \begin{cases} arg \ \ max \left\{ \left[\tau_{ij}(t)\right]^{\alpha} * \left[SUCC_{ij}(A_t)\right]^{\beta} * \left[\frac{1}{s_{ij}}\right]^{\delta} * \left[\frac{1}{H_{ij}}\right]^{\phi} \right\} & if \ q \le q_0 \\ J & if \ q > q_0 \end{cases} \quad (1)$$

where J is chosen according to the probability p_{ij}

$$p^{k}{}_{ij} = \frac{\left[\tau_{ij}(t)\right]^{\alpha} * \left[SUCC_{ij}(A_t)\right]^{\beta} * \left[\frac{1}{s_{ij}}\right]^{\delta} * \left[\frac{1}{H_{ij}}\right]^{\phi}}{\sum \left[\tau_{ij}(t)\right]^{\alpha} * \left[SUCC_{ij}(A_t)\right]^{\beta} * \left[\frac{1}{s_{ij}}\right]^{\delta} * \left[\frac{1}{H_{ij}}\right]^{\phi}} \quad (2)$$

3 Computational results

The benchmark problem set consists of two subsets : the first consists of eight instances, each with 15, 25, 35 and 45 jobs [4]. The second instance subset, taken from the work of [3], consists of eight instances each with 55, 65, 75 and 85 jobs. All the experiments were run on an Itanium with a 1.4 GHz processor and 4 GB RAM. Each instance was executed 20 times. Table 1 compares the results of different approaches. All the presented results show the average deviation to the best known solution for the sixty-four instances. The ACOG, ACOL and ACOS columns show the results of the ACO algorithm of [3], [5] and the presented algorithm in the Section 2, respectively. The HGA column presents the results of the hybrid genetic algorithm of [9] whose results are summarized in the last row. This latter algorithm represents the best approach found in the literature.

Table 1 Comparison of different approaches

Jobs	ACOG	ACOL	ACOS	HGA
15	0,8	0,6	0,0	0,0
25	2,3	3,1	0,0	0,0
35	18,0	25,0	2,3	0,0
45	5,9	4,8	2,1	0,8
55	6,5	5,1	3,5	1,2
65	150,4	136,0	58,4	18,4
75	6,4	5,4	3,6	1,1
85	9,9	7,6	4,3	2,6

The first observation is that the introduced ACO algorithm ACOS significantly improves all of the ACO results, especially for larger instances (the 65 jobs instance). Indeed, the use of the archive and the look-ahead information have been shown to improve solution quality, also enhancing the relative and the absolute order. Also, the integration of the local search allows the ACOS algorithm to show good results compared to the HGA algorithm.

4 Conclusion

In this work, we have introduced an ACO algorithm that integrates look ahead informations and archive concepts in the transition rule to minimize the total tardiness in the single machine problem with sequence-dependent setup times. The proposed approach is essentially based on adapting the transition rule to the specificities of the studied problem. The numerical experiments allowed us to demonstrate the efficiency of our approach to this problem.

An extension of this work would be to use this approach for other scheduling problems in particular and other optimization problems in general, especially real-world problems.

References

1. A. Allahverdi and 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, 985-1032, 187, 3 (2008)
2. G. Conner : 10 Questions. Manufacturing Engineering Magazine, 93-99 (2009)
3. C. Gagné, W.L. Price, M. Gravel: Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. Journal of the Operational Research Society, 53, 895-906 (2002)
4. G. L. Ragatz : A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence dependent setup times. Proceedings twenty-fourth annual meeting of the Decision Sciences Institute, 1375-1377 (1993)
5. C. Liao and H. Juan : An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers and Operations Research, 1899-1909, 34 (2007)
6. M. Pinedo : Scheduling Theory: Algorithm and Systems.Prentice-Hall (2002)
7. R. L. Graham, E. L. Lawler, J. K. Lenstra and A. G. H. Rinnooy Kan : Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics,287-326,5 (1979)
8. J. Du and J. Y. T. Leung : Minimizing total tardiness on one machine is NP-Hard. Mathematics and Operations Researchs, 15, 438-495 (1990)
9. A.Sioud, M. Gravel, C. Gagné : A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times. Computers and Operations Research,2415-2424, 39 (2012)
10. I. Or : Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. Ph. D Thesis, Northwestern University, Illinois (1976)

MISTA 2013

Algorithm performance prediction for nurse rostering

Tommy Messelis · Patrick De Causmaecker · Greet Vanden Berghe

Abstract In this paper, we investigate accurate performance prediction models for nurse rostering algorithms. The study is based on results of a similar approach for formally stated decision problems. Due to the complex nature of nurse rostering problems, we are bound to use heuristic methods that deliver good solutions in a reasonable amount of time. There is no guarantee of solutions being optimal. In practice, it is not even possible to employ exhaustive search methods on realistic problem instances. We quantify the performance of an algorithm as the quality of the obtained solution, after a predetermined amount of computation time.

We build prediction models based on simple characteristics of the instances. We develop a feature set that characterises the instances in terms of simple values. Such an approach allows for fast and accurate predictions. The application of the methodology to nurse rostering has not been attempted before.

1 Introduction

The nurse rostering problem is an NP-hard combinatorial optimisation problem with origins in an application of operational research (Warner, 1976). It is in essence the problem of finding an assignment of a number of nurses to a set of shifts in a hospital ward, while satisfying a number of hard and soft constraints. Hard constraints must be satisfied for a schedule to be valid (e.g. a nurse can not work in two places at the same time), soft constraints can be violated at a certain cost. The aim is to find a valid schedule with an as low as possible cost. Constraints are often contradictory. It is in practical circumstances almost always impossible to find a schedule that violates no (soft) constraints at all.

In literature, many different versions of the nurse rostering problem exist. Depending on the country, the region, or even the specific wards within one hospital, other types of constraints are present. Hence, there is no general definition of *the* nurse rostering problem.

Tommy Messelis, Patrick De Causmaecker
CODeS Research Group, member of iMinds-itec-KU Leuven
E. Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: {tommy.messelis,patrick.decausmaecker}@kuleuven-kulak.be

Greet Vanden Berghe
CODeS Research Group, Computer Science KAHO
Gebr. De Smetstraat 1, 9000 Gent, Belgium
E-mail: greet.vandenberghe@kahosl.be

Instead, there are many problem formulations, some more generally applicable than others. But not only the formulations differ; there exist at least as many solution methods as there are problem formulations. Burke et al (2004) give an overview of the state-of-the-art in nurse rostering in 2004. More recent publications on solving (a variant of) nurse rostering problems include, among others, Bard and Purnomo (2005); Vanhoucke and Maenhout (2009); Beddoe et al (2009); Brucker et al (2010); Bilgin et al (2012a). De Causmaecker and Vanden Berghe (2011) present a categorisation of nurse rostering problems along the lines of the $\alpha|\beta|\gamma$ notation for scheduling (Graham et al, 1979)[1].

Performance prediction for practical optimisation problems is a fairly new topic in operational research. It finds its roots in solution methods for formally stated problems and is based on the notion of an empirical hardness model (Leyton-Brown et al, 2002; Nudelman et al, 2004). An empirical hardness models is a mapping from instance properties onto a hardness measure. Such models are used to predict the hardness of a specific instance for a certain solution method. The hardness is called *empirical* because in this context, it is inherently linked to the algorithm that is used to solve the problem. The empirical hardness is thus a combination of both the intrinsic hardness of an instance, and the ability of an algorithm to tackle this instance. Leyton-Brown et al (2002) propose a method for building such prediction models based on linear regression techniques. Their aim is to predict the running time of complete search methods for the winner determination problem in combinatorial auctions. Nudelman et al (2004) have applied these ideas to propositional satisfiability problems. We will investigate performance prediction in the context of nurse rostering. In this specific context, domain expert knowledge can be used to generate candidate features. We investigate the effectiveness of these domain related features. Instead of using mainly linear regression techniques, we will also explore other machine learning techniques for building empirical hardness models. A similar study has been done for the university timetabling problem by Kostuch and Socha (2004). Almajano et al (2010) apply the same ideas to build an empirical hardness model for predicting the running time of an algorithm for the winner determination problem in mixed multi-unit combinatorial auctions. The work of Smith-Miles and Lopes (2011) is related to performance predictions. They investigate the boundaries of good algorithm performance in instance space for job-shop scheduling problems. They employ self-organising maps to visualize the regions in instance space where an algorithm performs well.

For most heuristic algorithms for nurse rostering, running time is a parameter and not a measure for the complexity of the problem at hand. Typically, the algorithms are allowed to run for a certain amount of time, after which the best found solution is reported. As a consequence, and in contrast to formally stated decision problems, predicting the running time will not be our aim. We will try to predict the quality of the solution produced by an algorithm under given circumstances (like e.g. the running time). The quality attributed to a specific solution is generally measured by an objective function. The algorithms set out to optimise this function. Heuristic algorithms in general produce suboptimal solutions. Circumstances are such that this is often the best you can get. Our aim will be to predict the value of this function based on a limited number of efficiently computable instance features. The objective function for nurse rostering problems is typically expressed as a weighted sum of constraint violations. The resulting quality, measured by this objective function, is hence related to the intrinsic complexity of the constraint satisfaction problem and hardness features for this problem can be expected to be relevant. On the other hand, certain algorithms may be better fit than others to tackle specific difficulties in the problem setting, and hence,

[1] Also see `http://www.informatik.uni-osnabrueck.de/knust/class/`

one algorithm can be expected to perform well on one instance, while under-performing on other instances. This may even differ for different running times. The resulting quality is thus also dependent on the algorithm that is used and its configuration (e.g. parameter settings, stopping criteria, etc.). The solution quality is the most commonly used performance criterion for heuristic procedures. There exist other measures for the performance of an algorithm as well. Sometimes, heuristic procedures can provide information on the upper or lower bounds of the solution quality. Such bounds could be used as performance criterion. Alternatively, one could look at the time needed to find a solution within a predetermined bound. The algorithms in this study do not provide such bounds. Nevertheless, the solution quality is an important and commonly used property on which algorithms are compared in practice.

The paper is structured as follows. In the next section, we give a formulation of the nurse rostering problem discussed in this paper. Section 3 discusses the experimental setting. The results of the experiments, and a number of accurate prediction models for the quality of the solutions produced by recent solution methods, are presented in Section 4. In Section 5, we give a number of possible applications where this research can prove relevant, after which conclusions are drawn in Section 6.

2 The nurse rostering problem

The nurse rostering problem (NRP) considered in this study is following the formulation that was used in the First International Nurse Rostering Competition (2010 INRC, see Haspeslagh et al, 2010). We refer the reader to Appendix A for a detailed mathematical model of the problem formulation and give hereunder an overview of the ingredients of the problem.

The nurse rostering problem is essentially a problem of assigning nurses to a set of shifts that have to be covered. The difficulty of the problem lies in the constraints that have to be satisfied. Some constraints are considered hard and other constraints are soft. For a solution to be feasible, all hard constraints must be satisfied. An optimal solution is a feasible solution that satisfies as many soft constraints as possible. Soft constraints can also be weighted. In that case, an optimal solution minimizes the weighted sum of constraint violations. More specifically, a nurse rostering problem consists of the following:

- planning horizon: This is a number of consecutive days during which the nurses need to be assigned to shifts. The planning horizon can be of arbitrary size.
- shift types: Each day is divided into a number of (possibly overlapping) shift types. A shift type represents a time frame and specifies a skill set. (Nurses are required to have specific skills to be able to work certain shifts[2].)
- coverage constraints: These are the personnel requirements. For each shift type on each day, the number of required nurses is specified.
- nurse pool: A set of nurses in the hospital ward. Nurses have their own characteristics which are described by their skill set, their contract regulations and their specific requests.
- skill types: Each nurse can have multiple skills. Nurses are allowed to work shift types for which they own all the required skill types.
- contracts: A nurse works according to exactly one contract. A contract specifies all the constraints (including the corresponding weights) that a personal schedule should satisfy. Each contract consists of the following constraints:

[2] For example: there might be a `Day` shift type that requires a regular nurse, and a `DayHead` shift type that requires a head nurse.

  - maximum and minimum number of shift assignments during the scheduling period
  - maximum and minimum number of consecutive working days
  - maximum and minimum number of consecutive free days
  - maximum and minimum number of consecutive working weekends
  - maximum number of working weekends per four weeks
  - whether or not the nurse should have at least two days off after a series of night shifts
  - whether or not the working weekends should be complete (i.e. if the working weekends should be complete, then either a nurse works full weekends or not at all during the weekend)
  - whether or not the nurse should work identical complete weekends (i.e. whether or not he or she should work the same shifts on all days within a weekend)
  - a set of unwanted patterns (e.g. a nurse does not want to work an early shift directly after a late or night shift)
- individual requests: A nurse can request to (not) work a particular shift on a certain day, or any shift during a day. These are preferences of the individual nurses.
- global constraints (valid for all nurses):
  - single assignment constraint: This is a global constraint that specifies whether or not nurses should work at most 1 shift type per day.
  - alternative skill: This constraint specifies whether or not nurses are entitled to work shift types that require skill types for which they do not qualify.

Patterns can be of any form. Two types of patterns were present in the 2010 INRC. The first pattern type specifies a series of consecutive shift type assignments that should always be avoided (i.e. the pattern can start at any day in the planning horizon). The second pattern type specifies that a nurse should not have a free day before working any of a specific number of consecutive days (e.g. when a nurse has to work either on Saturday or Sunday, then she should also work on Friday). Other arbitrary patterns are also possible, but are not considered in this study.

3 Experimental setting

In order to build accurate prediction models, a sufficient amount of training data is required. As with any learning approach, it is important to clearly state the boundaries of the instance distribution for which predictions are to be made. We have built a random instance generator that produces instances similar to those of the 2010 INRC, which are inspired by real-world instances. This problem instance distribution is described by $SA2|V5|P$ in the classification of De Causmaecker and Vanden Berghe (2011). We consider different sequence constraints (S) and availabilities (A) with a fluctuating coverage (V) in a 5-shift structure. The shifts in the problem are labelled `Day`, `Early`, `Late`, `Night` and `DayHead`. The `DayHead` shift type requires a nurse with two skill types, the regular 'nurse' skill type and the 'head nurse' skill type. The other shift types require nurses that have the regular 'nurse' skill type. The coverage constraints are hard, the sequence constraints and availabilities can be violated at a certain cost. The planning horizon of the instance distribution is either two or four weeks (with uniform probability). The coverage constraints are randomly chosen per week and differ for week and weekend days. The required number of nurses per shift type are random values in appropriate intervals, similar to the medium track of the 2010 INRC. The number of nurses is varied between 20 and 40, both inclusive. Nurses can have multiple skill

types. In this distribution, all nurses have the standard 'nurse' skill type, while a number of nurses also have the additional 'head nurse' skill type. Each nurse works according to one out of four prototypical contracts. Each contract specifies all the sequence constraints on the personal schedules of the nurses. Contracts represent whether nurses work full-time, part-time, 75% or 80%. The constraint values in the contracts have been randomly chosen from appropriate intervals. Each nurse has a random number of `DayOn`, `DayOff`, `ShiftOn` and `ShiftOff` requests. All the contract constraints and requests have been given random weights.

The generator is sampled to construct a dataset of 800 random instances. This set is further on randomly partitioned into a training set of 600 instances, and a validation set of 200 instances. In what follows, we will build performance prediction models using the 600 training instances. The best models are furthermore evaluated on the validation set of unseen instances, in order to assess their true performance.

In order to be able to quickly make predictions on new instances, we need a feature set that characterises the nurse rostering instances. These features must be efficiently computable. We have designed such a set and describe it in detail in Appendix B. Calculating the feature values for a given instance can be done in polynomial time. It is merely a matter of iterating over the problem description and counting certain properties. The set consists of 305 distinct features, which can be categorized into five groups:

- *features related to the problem size:*
  - concerning the scheduling period (3)
  - concerning the workforce (3)
- *feature related to the coverage requirements* (17)
- *features related to the workforce structure* (15)
- *features related to the contract constraints:*
  - concerning constraint values (45)
  - ratios (minimum over maximum constraint values) (12)
  - special ratios concerning the tightness of constraints (9)
  - concerning boolean constraints (4)
  - concerning the occurrence of patterns in a contract (16)
- *features related to the requests* (56)

Since soft constraints can be weighted, we include a weighted and an unweighted variant of features concerning such constraints.

As argued above, the quality of a solution depends on the combination of the properties of the instance (i.e. the intrinsic hardness) and the algorithm that is used. In this study, we focus on two of the top-five competitors in the 2010 INRC, and we build prediction models for both algorithms. The choice of algorithms is arbitrary although we require the availability of a UNIX executable. Both algorithms are local search metaheuristics. We consider the algorithms as black-boxes. They will be denoted as `Algorithm A` and `Algorithm B` in what follows.

When building prediction models, a considerable amount of data must be available. On all instances in both the training and the validation set, both algorithms were run five times each and their average performance is recorded. The stopping criterion is set to ten minutes of calculation time, as in the medium track of the 2010 INRC. We have parallellized the execution of this task and employed the cluster infrastructure of the VSC - Flemish Supercomputer Center[3].

[3] The VSC is funded by the Hercules foundation and the Flemish Government - department EWI. More information at `https://www.vscentrum.be/`

Table 1 Correlation coefficients of the various models.

		correlation coefficients (R)		
		training set	cross-validation	validation set
`Algorithm A`	Linear Regression	0.965	0.948	0.955
	Grid Search	0.958	0.942	0.956
	M5P tree	0.994	**0.988**	0.992
	Decision Table	0.978	0.963	0.959
	M5 Rules	0.994	0.985	0.989
`Algorithm B`	Linear Regression	0.967	0.950	0.957
	M5P tree	0.994	**0.990**	0.994
	Decision Table	0.980	0.964	0.959
	M5 Rules	0.993	0.981	0.991

Before the actual model construction can begin, a preprocessing step is performed to delete features that are uni-valued[4] or perfectly correlated to other features. The feature set was developed for general nurse rostering problems, given the formulation as presented in Section 2. Our instance distribution however poses some limits on a number of concepts. Examples are the shift types and skill types. There are always five different shift types and two different skill types present. Features concerning the variation of these concepts are thus uni-valued and hence removed. As a result, 129 potentially useful features remain.

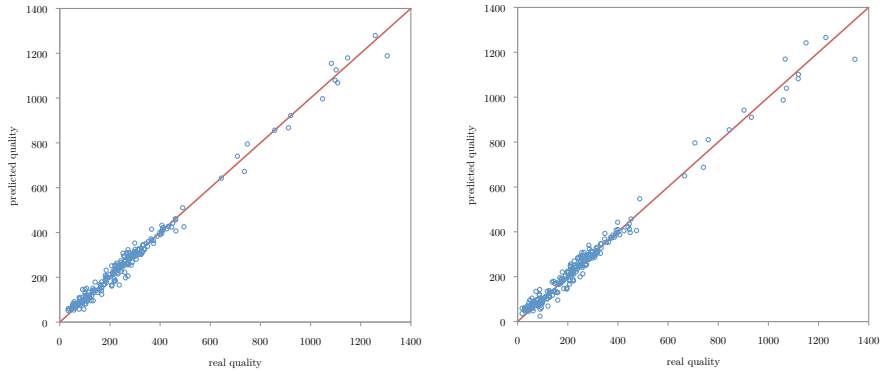
4 Results

There are a number of possible approaches to build mappings from the feature space onto the performance criterion. The WEKA software tool (Hall et al, 2009) provides an extensive collection of machine learning and data mining techniques enabling the construction of such mappings. We experiment with a number of machine learning techniques, in order to find the most accurate model for predicting the algorithm performance. Prediction models are learned on the training set and evaluated using 10-fold cross-validation on the training set. This evaluation technique gives an indication of how well the prediction model will perform on unseen instances. The best models are further on evaluated on the validation set, in order to determine their true performance on unseen instances.

We start with building prediction models using all of the 129 available features. Table 1 summarizes the results of various machine learning techniques in WEKA. It shows the correlation coefficients of several models, for both algorithms. High correlation coefficients indicate good predictive power of the models. We see that for both algorithms, the M5P tree model achieves the best correlation, based on cross-validation on the training set. An M5P tree model is a regression tree, i.e. a decision tree with regression models at the leaves. Based on the values of certain features, one of several regression models is used to predict the solution quality. We refer to Quinlan (1992) and Wang and Witten (1997) for more details on the M5P tree algorithm in WEKA. The correlation coefficients of the regression trees are $R = 0.988$ and $R = 0.990$ for the models predicting the quality of the solutions of `Algorithm A` and `Algorithm B` respectively. Table 1 also shows the correlation coefficients of the models when evaluated on the validation set of unseen instances. The resulting correlation coefficients of the regression tree models on the validation set are $R = 0.992$ and $R = 0.994$ for `Algorithm A` and `Algorithm B` respectively. This shows that the models can indeed accurately predict the performance of both algorithms on unseen instances. Figure 1

[4] having the same feature value for all instances

Fig. 1 Real performance versus predicted performance on the validation set.



(a) `Algorithm A` (b) `Algorithm B`

shows a plot of the actual performance of the algorithms versus the predicted performance on the validation set of unseen instances, (a) for `Algorithm A` and (b) for `Algorithm B`. All data points lie close to the diagonal, which confirms that predictions are highly accurate.

It is a common understanding in the machine learning community that smaller models should be preferred over larger models. Using fewer features and thus a simpler model reduces the risk of over-fitting. WEKA offers several methods for feature selection, both learner-independent and learner-dependent techniques. The main difference is that learner-dependent techniques select features during the model construction phase, while learner-independent techniques select a feature set prior to the actual model construction. Learner-dependent techniques select features in an iterative way, either forward, backward or bi-directional. Forward selection starts with building a model that uses only one feature, the feature most correlated to the performance criterion. Other features are iteratively added to the model until there is no more (significant) improvement on accuracy. Backward feature selection is the reverse process and starts with a model containing all features. Features are then iteratively removed until the accuracy drops significantly. The bi-directional process can either add or remove features at each step. The main disadvantage of such a technique is that many prediction models must be built at each step. Depending on the type of model, this can require a lot of time, especially when many features are available. Learner-independent techniques generally require less time. Such techniques select a feature set based on data properties (like e.g. feature correlation). This can also be an iterative process. In this study, we use a forward and backward correlation-based feature selection procedure offered by WEKA to reduce the feature set before the model construction. For both algorithms, 21 features remain. It appears that the most important features are related to the coverage constraints, contract constraints and the requests of the individual nurses:

- *features related to the coverage requirements:* (5)
  - the minimum number of required nurses per day
  - the standard deviation, variation and maximum of the number of required nurses per shift
  - the number of nurses divided by the number of shifts that need to be covered

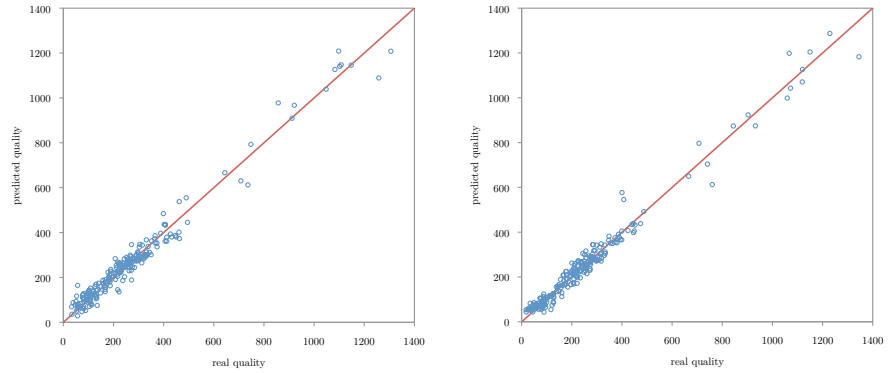
Table 2 Correlation coefficients of the various models after feature selection.

		correlation coefficients (R)		
		training set	cross-validation	validation set
`Algorithm A`	Linear Regression	0.763	0.740	0.729
	M5P tree	0.990	**0.985**	0.986
	Decision Table	0.973	0.965	0.972
	M5 Rules	0.988	0.978	0.982
	Multilayer Perceptron	0.990	0.960	0.963
`Algorithm B`	Linear Regression	0.759	0.735	0.721
	M5P tree	0.989	**0.984**	0.986
	Decision Table	0.972	0.963	0.971
	M5 Rules	0.989	0.977	0.982
	Multilayer Perceptron	0.990	0.955	0.960

- *features related to the contract constraints:* (13)
  - concerning constraint values:
    - the maximum number of assignments: maximum over all contract types
    - the minimum number of assignments: standard deviation over all contract types
    - the maximum number of consecutive working days: variation over all contract types
    - the minimum number of consecutive working weekends: minimum over all contract types
    - the maximum number of consecutive free days: maximum over all contract types
    - the minimum number of consecutive free days: mean, variation and minimum over all contract types
  - ratios:
    - the maximum over the minimum number of assignments: maximum over all contract types
    - the maximum over the minimum number of consecutive working days: maximum over all contract types
    - the maximum over the minimum number of consecutive working weekends: mean over all contract types
    - the maximum over the minimum number of consecutive free days: mean over all contract types
  - special ratios concerning the tightness of constraints:
    - the maximum number of assignments over the minimum number of consecutive working days: maximum over all contract types
- *features related to the requests:* (3)
  - the number of `Off` requests (both `dayOff` and `shiftOff`): mean, standard deviation and variation over the shifts

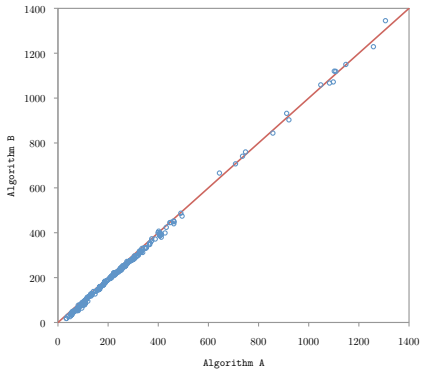
The correlation coefficients of several performance prediction models based on this reduced feature set are given in Table 2. As for the larger models, the M5P tree models produce the most accurate results, achieving correlation coefficients near those of Table 1. The correlation of the models on the validation set of unseen instances is also very high. Figure 2 confirms this accurate predictive power. It shows a plot of the actual performance of the algorithms versus the predicted performance on the validation set of unseen instances, (a) for `Algorithm A` and (b) for `Algorithm B`. Although the feature selection process has significantly reduced the feature set, the prediction accuracy is only marginally lowered and not significantly worse. Tables 1 and 2 and the graphs in Figures 1 and 2 show that there is

Fig. 2 Real performance versus predicted performance on the validation set, after feature selection.



(a) `Algorithm A` (b) `Algorithm B`

Fig. 3 Performance of `Algorithm A` versus `Algorithm B`.



no clear distinction between the larger and smaller models in terms of accuracy, the smaller models should thus be preferred.

Comparing the results for both algorithms, we find that the same features are being selected for building the smaller models. This is not very surprising, after all, both algorithms perform similar. Figure 3 shows a plot of the performance of `Algorithm A` versus the performance of `Algorithm B`. We see that these performances are strongly correlated ($R = 0.999$). For most instances, `Algorithm A` performs a little bit better than `Algorithm B` (most data points are just below the diagonal). This explains why the same features are important for both algorithms.

When we look at the models in more detail, it appears that some features (in the reduced feature set) are still more important than others. An outtake of the M5P tree model for `Algorithm A`, based on the reduced feature set, is displayed in Figure 4. We see that the decision tree is split at the highest levels using the `nrNursesPerShift` feature. This feature represents the number of nurses divided by the total number of shifts that need to be worked during the planning horizon. Moreover, the regression models at the leafs have generally higher coefficients for this feature than for other features, indicating that this is a

Fig. 4 Outtake of the M5P tree model for `Algorithm A` (reduced feature set).

```
nrNursesPerShift <= 0.053 :
|   nrNursesPerShift <= 0.038 : LM1
|   nrNursesPerShift >  0.038 :
|   |   nrNursesPerShift <= 0.049 : LM2
|   |   nrNursesPerShift >  0.049 : LM3
nrNursesPerShift >  0.053 :
|   nrNursesPerShift <= 0.071 :
|   |   requiredNursesPerShift_stddev <= 3.244 :
|   |   |   MinConsecutiveFreeDays_mean <= 4.338 : LM4
|   |   |   MinConsecutiveFreeDays_mean >  4.338 : LM5
|   |   requiredNursesPerShift_stddev >  3.244 :
|   |   |   requestsOffPerShift_stddev <= 0.975 : LM6
|   |   |   requestsOffPerShift_stddev >  0.975 :
|   |   |   |   ratio_MaxOverMinConsecutiveFreeDays_mean <= 1.477 :
|   |   |   |   |   requiredNursesPerDay_minimum <= 21.5 : LM7
|   |   |   |   |   requiredNursesPerDay_minimum >  21.5 : LM8
|   |   |   |   ratio_MaxOverMinConsecutiveFreeDays_mean >  1.477 :
|   |   |   |   |   MinConsecutiveFreeDays_mean <= 3.192 : LM9
|   |   |   |   |   MinConsecutiveFreeDays_mean >  3.192 :
|   |   |   |   |   |   ratio_MaxOverMinConsecFreeDays_mean <= 2.131 : LM10
|   |   |   |   |   |   ratio_MaxOverMinConsecFreeDays_mean >  2.131 : LM11
|   nrNursesPerShift >  0.071 :
|   |   requiredNursesPerShift_stddev <= 2.664 : LM12
|   |   requiredNursesPerShift_stddev >  2.664 : LM13

LM num: 1
quality =
        4580.2305   *  nrNursesPerShift
        + 22.3284   *  requiredNursesPerDay_minimum
        + 17.0092   *  requiredNursesPerShift_stddev
        + 5.8534    *  requiredNursesPerShift_maximum
        + 1.7355    *  MaxNumAssignments_maximum
        - 10.6989   *  MinNumAssignments_stddev
        + 1.148     *  MaxConsecutiveFreeDays_maximum
        + 12.2018   *  MinConsecutiveFreeDays_mean
        - 125.4501  *  MinConsecutiveFreeDays_variation
        + 1.6656    *  MinConsecutiveFreeDays_minimum
        + 2.4521    *  ratioMaxNumAssignmentsOverMinConsecWorkingDays_maximum
        - 85.204    *  requestsOffPerShift_mean
        - 35.0942   *  requestsOffPerShift_variation
        - 28.1872
<other regression models omitted>
```

very important factor for predicting the performance of the algorithms[5]. In order to further demonstrate its importance, we have deleted this feature from the reduced set and tried to build prediction models using only the resulting 20 features. As it turns out, none of the available methods is able to achieve a correlation coefficient higher than $R = 0.779$, using cross-validation on the training set as before. Prediction accuracy is poor for these models, proving the crucial importance of the `nrNursesPerShift` feature.

[5] This is only true when all features are normalized, which is the case here.

5 Relevance

We have shown in the previous section that we can build accurate prediction models for the performance of two state-of-the-art algorithms for the nurse rostering problem (in the formulation of the 2010 INRC). Here, we focus on how such prediction models can be applied in practice.

Predicting the output of algorithms on specific instances opens several perspectives for their application. The type of algorithm for which we build predictors in this paper is fit for real world applications. When looking at benchmarks, often no single algorithm outperforms all others. Competitive state-of-the-art algorithms have to fight on each instance. On many instances, several state-of-the-art algorithms perform similar to each other. In other cases, one algorithm seems to be better able to deal with specific characteristics of the instance than the other algorithms. This could very well be just the other way around for other instances. Each algorithm can thus have its own strengths and weaknesses. When computing resources are scarce, and instances need to be solved quickly, selecting the best algorithm for a given instance is the way to go. Performance prediction models can straightforwardly be employed for automatic algorithm selection tools. Based on the performance predictions of a set of algorithms, the best algorithm can be selected to run on a given instance.

Apart from algorithm selection, automated negotiation schemes require a cost estimate of a specific scenario. This estimate needs to be reliable but not necessarily very accurate. The negotiation scheme should not take long to evaluate. In the case of nurse rostering, one can think of a system ruling the exchange of nurses between wards in a hospital or from an exchange pool. When considering calling in a nurse from the pool, internal rearrangements normally produce several possibilities. These possibilities are negotiated among the wards taking into account the local cost/benefit as expressed by the objective function. It is not always feasible to exactly evaluate all the possibilities, as this involves solving many nurse rostering instances. Fast estimates of the solution quality that an algorithm can achieve are needed. Performance prediction models can provide such estimates (Haspeslagh et al, 2007, 2013; Haspeslagh and De Causmaecker, 2013).

Another application can be found in patient admission scheduling (Bilgin et al, 2012b). An optimal patient admission policy is essential for the financial sanity of hospitals and it impacts the performance indicators used in hospital evaluation. Assigning patients to specific rooms in a hospital influences the coverage requirements of the wards and the workloads of the nurses. It can thus be useful to take into account these effects when scheduling patient admissions. A reliable predictor for algorithm performance can support fast decision making in patient admission and could increase the hospital's overall capacity utilisation.

6 Conclusions

In this paper, we have investigated the construction of accurate performance prediction models for two state-of-the-art algorithms for nurse rostering. We have built such models using two approaches. We started by using all available features, resulting in high-quality prediction models. We further on applied feature selection techniques to reduce the feature set. The models bases on the reduced feature set achieve similar results as the larger models. Feature selection has thus no negative impact on the prediction accuracy. It furthermore raises the potential for domain experts to try and understand the resulting models.

It is important to note that the performance prediction models are valid within the boundaries of the instance distribution that is used to train the models. A necessary precondition

is the existence of a large enough set of training data. Such a set was not yet available. We have therefore built an instance generator that produces random instances that are similar to the instances used in the 2010 International Nurse Rostering Competition. This generator varies the properties of the instances around appropriate values like those used in the medium track of the competition. It chooses both longer and shorter scheduling periods and varies the number of nurses. The coverage requirements are chosen so that realistic scenarios are built. This instance generator is used to build a large data set, which is further partitioned into a training set and a validation set.

In order to enable quick performance predictions, we need a way to quickly characterized problem instances as vectors of feature values. We have therefore designed an efficiently computable feature set that summarizes generic nurse rostering instances by the means of 305 distinct feature values. This is an important contribution that is essential for the success of the prediction models.

Using a high performance computing cluster, we ran two state-of-the-art algorithms on all generated problem instances. Afterwards, we experimented with several machine learning techniques, resulting in the construction of highly accurate prediction models for the performance of both algorithms.

These prediction models can lead to significant advances in the performance of applications. An automatic algorithm selection tool, using a portfolio of state-of-the-art algorithms could further improve the overall performance of these algorithms. Such a tool can employ performance prediction models as a basis for the selection of the best algorithm to be applied. Automated negotiation schemes for the exchange of personnel in a hospital can benefit from our prediction models supplying quick quality approximations. Further away, but also as a future direction, patient admission scheduling systems can employ performance prediction models. Assigning patients to beds in specific wards influences the coverage constraints in those wards. Taking these effects into account can lead to better decisions. As a first step towards such applications, we want to broaden the scope of this study by including other algorithms and instance distributions.

References

Almajano P, Cerquides J, Rodriguez-Aguilar JA (2010) Empirical hardness for mixed auctions. In: Current Topics in Artificial Intelligence, Springer, pp 161–170

Bard JF, Purnomo HW (2005) Preference scheduling for nurses using column generation. European Journal of Operational Research 164:510–534

Beddoe G, Petrovic S, Li J (2009) A hybrid metaheuristic case-based reasoning system for nurse rostering. Journal of Scheduling 12:99–119

Bilgin B, De Causmaecker P, Rossie B, Vanden Berghe G (2012a) Local search neighbourhoods for dealing with a novel nurse rostering model. Annals of Operations Research 194:33–57

Bilgin B, Demeester P, Misir M, Vancroonenburg W, Vanden Berghe G (2012b) One hyperheuristic approach to two timetabling problems in health care. Journal of Heuristics 18(3):401–434

Brucker P, Burke EK, Curtois T, Qu R, Vanden Berghe G (2010) A shift sequence based approach for nurse scheduling and a new benchmark dataset. Journal of Heuristics 16(4):559–573

Burke EK, De Causmaecker P, Vanden Berghe G, Van Landeghem H (2004) The state of the art of nurse rostering. Journal of Scheduling 7(6):441–499

De Causmaecker P, Vanden Berghe G (2011) A categorisation of nurse rostering problems. Journal of Scheduling 14(1):3–16

Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer PL, Johnson EL, Korte BH (eds) Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium, Elsevier, Annals of Discrete Mathematics, vol 5, pp 287–326

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The WEKA data mining software: an update. SIGKDD Explorations Newsletter 11:10–18

Haspeslagh S, De Causmaecker P (2013) Bridging the gap between short and mid term nurse rostering through a negotiation protocol, submitted

Haspeslagh S, De Causmaecker P, Vanden Berghe G (2007) Distributed decision making in hospital wide nurse rostering problems. In: Baptiste P, Kendall G, Munier-Kordon A, Sourd F (eds) Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2007), August 28-31 2007, Paris, France, pp 192–199

Haspeslagh S, De Causmaecker P, Stølevik M, Schaerf A (2010) First international nurse rostering competition 2010. Tech. rep., K.U.Leuven Campus Kortrijk, http://www.kuleuven-kortrijk.be/codes/

Haspeslagh S, Messelis T, Vanden Berghe G, De Causmaecker P (2013) An efficient translation scheme for representing nurse rostering problems as satisfiability problems, to appear in Proceedings of the 5th International Conference on Agents and Artificial Intelligence, February 2013, Barcelona, Spain

Kostuch P, Socha K (2004) Hardness prediction for the university course timetabling problem. In: Evolutionary Computation in Combinatorial Optimization, Springer, pp 135–144

Leyton-Brown K, Nudelman E, Shoham Y (2002) Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In: Constraint Programming (CP), pp 556–572

Nudelman E, Leyton-Brown K, Devkar A, Shoham Y, Hoos H (2004) Understanding random SAT: Beyond the clauses-to-variables ratio. In: International Conference on Principles and Practice of Constraint Programming (CP), pp 438–452

Quinlan JR (1992) Learning with continuous classes. In: Proceedings of the 5th Australian joint Conference on Artificial Intelligence, Singapore, vol 92, pp 343–348

Smith-Miles K, Lopes L (2011) Generalising algorithm performance in instance space: A timetabling case study. In: Coello Coello CA (ed) Selected Papers of the 5th Learning and Intelligent Optimization Conference (LION 5), January 17-21 2011, Rome, Italy, Springer, Lecture Notes in Computer Science, vol 6683, pp 524–538

Vanhoucke M, Maenhout B (2009) On the characterisation and generation of nurse scheduling problem instances. European Journal of Operational Research 196(2):457–467

Wang Y, Witten IH (1997) Inducing model trees for continuous classes. In: Poster Papers of the 9th European Conference on Machine Learning (ECML 97), Prague, Czech Republic, pp 128–137

Warner DM (1976) Scheduling nursing personnel according to nursing preference: A mathematical programming approach. Operations Research 24(5):842–856

A Mathematical model

We present an exact description of the nurse rostering problem considered in this work as a mixed integer program. The objective is to minimize the following sum (1) subject to the linear expressions in (2 - 32).

$$\begin{aligned}\text{MINIMIZE}\{\ & V(T^{max})+V(T^{min})+V(CW^{max})+V(CW^{min})+V(CF^{max})+\\ & V(CF^{min})+V(CWW^{max})+V(CWW^{min})+V(CW)+V(CIW)+\\ & V(D^{off})+V(S^{off})+V(D^{on})+V(S^{on})+V(SK)+V(PAT^{cons})+V(PAT^{free})\ \}\end{aligned} \tag{1}$$

Let N be the set of nurses, D the set of days in the scheduling period, W the set of all weekends in the scheduling period, and S the set of shift types. Equation (2) defines the binary decision variables $x_{n,d,s}$. They state whether each nurse n is working shift type s on day d. A set of auxiliary variables $p_{n,d}$ is defined in (3). These variables indicate whether nurse n is working any shift type on day d. A similar set of auxiliary variables $q_{n,w}$ is defined in (4). These indicate whether nurse n is working any shift during weekend w. A weekend w is defined by its first day $d_{w,1}$ and its length k (which is equal for all weekends in the planning horizon)[6].

$$\forall n \in N, \forall d \in D, \forall s \in S : x_{n,d,s} = \begin{cases} 1 \text{ if nurse } n \text{ is assigned a shift type } s \text{ on day } d \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

$$\forall n \in N, \forall d \in D : -|S| p_{n,d} + \sum_{s \in S} x_{n,d,s} \leq 0 \text{ and } -p_{n,d} + \sum_{s \in S} x_{n,d,s} \geq 0 \tag{3}$$

$$\forall n \in N, \forall w \in W : -k q_{n,w} + \sum_{i=0}^{k-1} p_{n,d_{w,1}+i} \leq 0 \text{ and } -q_{n,w} + \sum_{i=0}^{k-1} p_{n,d_{w,1}+i} \geq 0 \tag{4}$$

There are two types of hard constraints, which are enforced by (5) and (6). Equation (5) defines the coverage requirement constraints. These constraints $C_{d,s}$ state the exact number of nurses required to work shift type s on day d. Inequality (6) defines the single assignment per day constraint. It states that nurses should work at most one shift type per day.

$$\forall d \in D, \forall s \in S : \sum_{n \in N} x_{n,d,s} = C_{d,s} \tag{5}$$

$$\forall n \in N, \forall d \in D : \sum_{s \in S} x_{n,d,s} \leq 1 \tag{6}$$

The other constraints in the problem are considered soft. Each nurse n has a weight W_n^C associated with each constraint type C. When violated, the constraints contribute to the objective function proportionally to the degree of violation, multiplied by the individual weight of the nurse which causes the violation. The objective function is represented in (1). The goal is to minimize a sum of weighted constraint violations, which are defined by the expressions in (7 - 32).

Constraints T_n^{max} and T_n^{min} on the total workload of nurses limit respectively the maximum and minimum number of shifts worked per nurse n. The total number of weighted constraint violations $V(T^{max})$ and $V(T^{min})$ are defined in (7) and (8).

$$V(T^{max}) = \sum_{n \in N} W_n^{T^{max}} max\{(\sum_{d \in D} \sum_{s \in S} x_{n,d,s}) - T_n^{max}, 0\} \tag{7}$$

$$V(T^{min}) = \sum_{n \in N} W_n^{T^{min}} max\{T_n^{min} - (\sum_{d \in D} \sum_{s \in S} x_{n,d,s}), 0\} \tag{8}$$

The constraints CW_n^{max} and CW_n^{min} limit the maximum and minimum number of consecutive working days for each nurse n. The total number of weighted constraint violations $V(CW^{max})$ and $V(CW^{min})$ are defined in (9) and (10), subject to the inequalities in (11) and (12). The auxiliary variables $t_{n,d,0}$ in (11) indicate whether nurse n works on day $d+1$ while he or she is free on day d. For $i > 0$, the auxiliary variables $t_{n,d,i}$ in (12) indicate whether there is a consecutive row of working days (of length i) for nurse n since day d. In counting these constraint violations, nurses are assumed to be free on days outside the planning period (i.e. $p_{n,\alpha} = 0$ for $\alpha \notin D$).

$$V(CW^{max}) = \sum_{n \in N} W_n^{CW^{max}} \sum_{d \in D} max\{(\sum_{i=0}^{CW_n^{max}} p_{n,d+i}) - CW_n^{max}, 0\} \tag{9}$$

[6] for example: a weekend w of $k = 2$ days consists of the consecutive days $d_{w,1}$ and $d_{w,2}$.

$$V(CW^{min}) = \sum_{n\in N} W_n^{CW^{min}} \sum_{d\in D} (CW_n^{min} t_{n,d,0} - \sum_{i=1}^{CW_n^{min}} t_{n,d,i}) \tag{10}$$

$$0 \le t_{n,d,0} \le 1 - p_{n,d} \text{ and } p_{n,d+1} - p_{n,d} \le t_{n,d,0} \le p_{n,d+1} \tag{11}$$

$$0 \le t_{n,d,i} \le p_{n,d+i} \text{ and } t_{n,d,i} \le t_{n,d,i-1} \tag{12}$$

Similarly, the constraints CF_n^{max} and CF_n^{min} limit the maximum and minimum number of consecutive free days for each nurse n. The total number of weighted constraint violations $V(CF^{max})$ and $V(CF^{min})$ are defined in (13) and (14), subject to the inequalities in (15) and (16). The auxiliary variables $t_{n,d,0}$ in (15) indicate whether nurse n works on day d while he or she is free on day $d+1$. For $i > 0$, the auxiliary variables $t_{n,d,i}$ in (16) indicate whether there is a consecutive row of free days (of length i) for nurse n since day d. In counting these constraint violations, nurses are assumed to be working on days outside the planning period (i.e. $p_{n,\alpha} = 1$ for $\alpha \notin D$).

$$V(CF^{max}) = \sum_{n\in N} W_n^{CF^{max}} \sum_{d\in D} max\{(\sum_{i=0}^{CF_n^{max}} 1 - p_{n,d+i}) - CF_n^{max}, 0\} \tag{13}$$

$$V(CF^{min}) = \sum_{n\in N} W_n^{CF^{min}} \sum_{d\in D} (CF_n^{min} t_{n,d,0} - \sum_{i=1}^{CF_n^{min}} t_{n,d,i}) \tag{14}$$

$$0 \le t_{n,d,0} \le p_{n,d} \text{ and } p_{n,d} - p_{n,d+1} \le t_{n,d,0} \le 1 - p_{n,d+1} \tag{15}$$

$$0 \le t_{n,d,i} \le 1 - p_{n,d+i} \text{ and } t_{n,d,i} \le t_{n,d,i-1} \tag{16}$$

The third type of consecutiveness constraints CWW_n^{max} and CWW_n^{min} limit respectively the maximum and minimum number of consecutive working weekends per nurse n. The total number of weighted constraint violations $V(CWW^{max})$ and $V(CWW^{min})$ are defined in (17) and (18), subject to the inequalities in (19) and (20). The auxiliary variables $t_{n,w,0}$ in (19) indicate whether nurse n works on weekend $w+1$ while he or she is free on weekend w. For $i > 0$, the auxiliary variables $t_{n,w,i}$ in (20) indicate whether there is a consecutive row of working weekends (of length i) for nurse n since weekend w. In counting these constraint violations, nurses are assumed to be free on weekends outside the planning period (i.e. $q_{n,\alpha} = 0$ for $\alpha \notin W$).

$$V(CWW^{max}) = \sum_{n\in N} W_n^{CWW^{max}} \sum_{w\in W} max\{(\sum_{i=0}^{CWW_n^{max}} q_{n,w+i}) - CWW_n^{max}, 0\} \tag{17}$$

$$V(CWW^{min}) = \sum_{n\in N} W_n^{CWW^{min}} \sum_{w\in W} (CWW_n^{min} t_{n,w,0} - \sum_{i=1}^{CWW_n^{min}} t_{n,w,i}) \tag{18}$$

$$0 \le t_{n,w,0} \le q_{n,w} \text{ and } q_{n,w} - q_{n,w+1} \le t_{n,w,0} \le 1 - q_{n,w+1} \tag{19}$$

$$0 \le t_{n,w,i} \le q_{n,w+i} \text{ and } t_{n,w,i} \le t_{n,w,i-1} \tag{20}$$

The complete weekend constraints $CW_n \in \{1,0\}$ specify whether or not nurse n should work either all days or no days at all during a weekend[7]. The total number of weighted constraint violations $V(CW)$ is defined in equation (21).

$$V(CW) = \sum_{n\in N} W_n^{CW} \sum_{w\in W} CW_n (kq_{n,w} - \sum_{i=0}^{k-1} p_{n,d_{w,1+i}}) \tag{21}$$

Similarly, the complete identical weekend constraints $CIW_n \in \{1,0\}$ specify whether or not nurse n should work the same shift type on all days in a complete weekend[8]. The total number of weighted constraint violations $V(CIW)$ is defined in equation (22), subject to the inequalities in (23). The auxiliary variables $t_{n,w,s}$ in (23) indicate whether nurse n is working shift type s during weekend w.

$$V(CIW) = \sum_{n\in N} W_n^{CIW} \sum_{w\in W} \sum_{s\in S} CIW_n (t_{n,w,s} k - \sum_{i=0}^{k-1} x_{n,d+i,s}) \tag{22}$$

$$-t_{n,w,s} k + \sum_{i=0}^{k-1} x_{n,d_{w,1+i},s} \le 0 \text{ and } -t_{n,w,s} + \sum_{i=0}^{k-1} x_{n,d_{w,1+i},s} \ge 0 \tag{23}$$

[7] $CW_n = 1$ if nurse n needs to work complete weekends, $CW_n = 0$ if she or he can work an arbitrary number of days in a weekend.

[8] $CIW_n = 1$ if nurse n should work identical days during a weekend, $CIW_n = 0$ if he or she can work different shift types during the days of a weekend.

Nurses can request to be free or working on specific days or shifts. Let D^{off} be the set of requests $req^{off}_{n,d}$ denoting that nurse n does not want to work on day d and D^{on} be the set of requests $req^{on}_{n,d}$ denoting that nurse n wants to work on day d. Let S^{off} be the set of requests $req^{off}_{n,d,s}$ denoting that nurse n does not want to work shift type s on day d and S^{on} be the set of requests $req^{on}_{n,d,s}$ denoting that nurse n wants to work shift type s on day d. The total number of weighted constraint violations $V(D^{off})$, $V(D^{on})$, $V(S^{off})$ and $V(S^{on})$ is defined in equations (24), (25), (26) and (27) respectively.

$$V(D^{off}) = \sum_{req^{off}_{n,d} \in D^{off}} W_n^{D^{off}} \sum_{s \in S} x_{n,d,s} \tag{24}$$

$$V(D^{on}) = \sum_{req^{on}_{n,d} \in D^{on}} W_n^{D^{on}} (1 - min\{1, (\sum_{s \in S} x_{n,d,s})\}) \tag{25}$$

$$V(S^{off}) = \sum_{req^{off}_{n,d,s} \in S^{off}} W_n^{S^{off}} x_{n,d,s} \tag{26}$$

$$V(S^{on}) = \sum_{req^{on}_{n,d,s} \in S^{on}} W_n^{S^{on}} (1 - x_{n,d,s}) \tag{27}$$

The alternative skill constraint $SK^{\text{alt}} \in \{0,1\}$ is common to all nurses. It specifies whether or not nurses can be assigned shift types for which they do not have all the required skill types[9]. The weighted number of constraint violations $V(SK)$ is defined in equation (28), subject to the inequalities in (29) and (30). SK is the set of skill types sk that are present in the problem, SK_s is the set of required skill types for shift type s and SK_n is the set of skill types of nurse n. The auxiliary variables $r_{n,s}$ in (30) denote whether nurse n has the required skill types to work shift s. The variables $b_{n,sk}$ denote whether nurse n has skill type sk.[10] Hence, the variables $t_{n,d,s}$ in (29) equal 1 if and only if nurse n works shift type s on day d, while not having all the required skill types.

$$V(SK) = SK^{\text{alt}}.W^{SK} \sum_{n \in N} \sum_{d \in D} \sum_{s \in S} t_{n,d,s} \tag{28}$$

$$t_{n,d,s} \geq x_{n,d,s} - r_{n,s} \tag{29}$$

$$-r_{n,s}|SK_s| + \sum_{sk \in SK_s} b_{n,sk} \geq 0 \text{ and } -r_{n,s} - |SK_s| + 1 + \sum_{sk \in SK_s} b_{n,sk} \geq 0 \tag{30}$$

Certain patterns in a nurses' schedule should be avoided. There are two types of patterns in the problem description. The first pattern type is a series of consecutive shifts (starting on any day) that should be avoided. Each nurse has a set PAT^{cons}_n of patterns pat that are unwanted. Each pattern pat is a sequence of shift types s_i and has a certain length, denoted by $length_{pat}$. The second pattern type avoids nurses to have a free day before working one of the successive days[11]. Similarly, each nurse has a set PAT^{free}_n of patterns pat that are unwanted. Each pattern pat now consists of a starting day d_{pat} (i.e. the day that should be free) and a certain length $length_{pat}$. The total number of weighted violations $V(PAT^{cons})$ and $V(PAT^{free})$ is defined in equations (31) and (32) respectively.

$$V(PAT^{cons}) = \sum_{n \in N} W_n^{PAT^{cons}} \sum_{d \in D} \sum_{pat \in PAT^{cons}_n} max\{((\sum_{s_i \in pat} x_{n,d+i,s_i}) - lentgh_{pat} + 1), 0\} \tag{31}$$

$$V(PAT^{free}) = \sum_{n \in N} W_n^{PAT^{free}} \sum_{pat \in PAT^{free}_n} max\{(min\{\sum_{i=1}^{lentgh_{pat}} p_{n,d_{pat}+i}, 1\} - (1 - p_{n,d_{pat}})), 0\} \tag{32}$$

[9] $SK^{\text{alt}} = 0$ if nurses are allowed to work shift types for which they do not have all the required skills, $SK^{\text{alt}} = 1$ otherwise.

[10] $b_{n,sk} = 1$ if $sk \in SK_n$ and $b_{n,sk} = 0$ otherwise.

[11] For example, a pattern could state that a nurse should not be free on the Friday preceeding a working weekend.

B NRP feature set

This appendix contains an extensive list of the NRP feature set developed for predicting algorithm performance for nurse rostering.

The 305 features in this set can be grouped into 5 categories: feature related to the problem size, features related to the coverage requirements, features related to the workforce structure, features related to the contract constraints and features related to the requests. All feature values can be calculated simply by iterating over the concepts in the problem description (i.e. in polynomial time). The calculation time for the instances considered in this study is a matter of a few milliseconds, which is negligible compared to the calculation time of the algorithms. The following list sums up all features. Whenever a concept can be weighted, the feature set includes both a weighted and an unweighted variant of the feature. The numbers between parentheses denote the number of unweighted features in the group.

- *features related to the problem size:* (6)
  - concerning the scheduling period:
    - the number of days in the planning period
    - the number of time units in the planning horizon
    - the number of different shift types
  - concerning the workforce:
    - the number of nurses
    - the number of distinct contract types
    - the number of skill types
- *features related to the coverage constraints:* (17)
  - the minimum, maximum, mean, standard deviation and variation of the number of required nurses
    - per day
    - per shift type
    - per skill type
  - the total number of shifts that need to be worked
  - the number of nurses divided by the total number of shifts that need to be worked
- *features related to the workforce structure:* (15)
  - the minimum, maximum, mean, standard deviation and variation of the number of
    - skill types per nurse
    - nurses per skill type
    - nurses per contract type
- *features related to the contract constraints:*
  - statistics on the constraint values: (45)
    the minimum, maximum, mean, standard deviation and variation over the contract types of:
    - the minimum number of assignments
    - the maximum number of assignments
    - the minimum number of consecutive working days
    - the maximum number of consecutive working days
    - the minimum number of consecutive free days
    - the maximum number of consecutive free days
    - the minimum number of consecutive working weekends
    - the maximum number of consecutive working weekends
    - the maximum number of working weekends in four weeks
  - ratios of constraint values: (12)
    the mean, minimum and maximum over the contract types of
    - the minimum-over-maximum ratio of
      - the number of assignments
      - the number of consecutive working days
      - the number of consecutive free days
      - the number of consecutive working weekends
  - special ratios concerning the tightness of the constraints: (9)
    the mean, minimum and maximum over the contract types of
    - the maximum number of assignments divided by the minimum number of consecutive working days
    - the minimum number of consecutive free days divided by the minimum number of consecutive working days
    - the number of shift types that can maximally be worked by the nurses (measured by their maximum number of assignments) divided by the total number of shifts that need to be worked

  - concerning boolean constraints: (4)
    the fraction of nurses that have the following boolean constraint set to true in their contract:
    - whether or not the working weekends should be complete
    - whether or not the nurse should work identical complete weekends
    - whether or not the nurse should have at least two days off after a series of night shifts
    - whether or nor the nurse should maximally work one shift a day
  - concerning the occurrence of patterns in a contract: (16)
    - the total number of patterns in the problem description
    - the minimum, maximum, mean, standard deviation and variation of the number of patterns per contract type
    - the minimum, maximum, mean, standard deviation and variation of the length of the patterns
    - the minimum, maximum, mean, standard deviation and variation of the weight of the patterns
- *features related to the requests:* (56)
  - the minimum, maximum, mean, standard deviation and variation over the nurses of
    - the total number of `On` requests (i.e. the sum of the number of `DayOn` and `ShiftOn` requests)
    - the total number of `Off` requests
    - the number of `DayOn` requests
    - the number of `DayOff` requests
    - the number of `ShiftOn` requests
    - the number of `ShiftOff` requests
  - the minimum, maximum, mean, standard deviation and variation over the days of
    - the number of `On` requests
    - the number of `Off` requests
  - the minimum, maximum, mean, standard deviation and variation over the shifts of
    - the number of `On` requests
    - the number of `Off` requests
  - the total number of
    - `On` requests
    - `Off` requests
    - `DayOn` requests
    - `DayOff` requests
    - `ShiftOn` requests
    - `ShiftOff` requests

MISTA 2013

Homogeneously non-idling schedules of unit-time jobs on identical parallel machines

Alain Quilliot · Philippe Chrétienne

Abstract In this paper, we study the basic homogeneous m-machine scheduling problem where weakly dependent unit-time jobs have to be scheduled within the time windows between their release dates and due dates so that, for any subset of machines, the set of the time units at which at least one machine is busy, is in interval. We first introduce the notions of pyramidal structure, k-hole, m-matching, preschedule, k-schedule and schedule for this problem. Then we provide a feasibility criteria for a preschedule. The key result of the paper is then to provide a structural necessary and sufficient condition for an instance of the problem to be feasible. We conclude by giving the directions of ongoing works and by bringing open questions related to different variants of the basic non-idling m-machine scheduling problem.

1 Introduction

Most scheduling problems assume that no cost is incurred when a machine waits between the completion of a job and the start of the next job. Morever, it is well-known that such waiting delays are often necessary to get optimality. This is the key feature why list algorithms, that do not allow a machine to wait for a more urgent job, do not generally provide optimal schedules. However, in some applications such as those described in [5], the cost of making a running machine stop and restart later is so high that a non-idling constraint is put on the machine so that only schedules without any intermediate delays are required. For example, if the machine is an oven that must heat different and not compatible pieces of work at a given high temperature, keeping the required temperature of the oven while it is empty may clearly be too costly. Problems concerning power management policies may also yield similar scheduling problems [6] where for example each idling period has a cost and the total cost has to be minimized

Alain Quilliot
Université Blaise Pascal, CNRS LIMOS, France
E-mail: alain.quilliot@isima.fr

Philippe Chrétienne
Université Paris 6, CNRS LIP6, France
E-mail: Philippe.Chretienne@lip6.fr

[1]. Note that the non-idling constraint will not necessarily ensure full machine utilization but will remove the cost of machine re-starts, maybe at the price of processing the jobs later.

Contrary to the well-known no-wait constraint in shop scheduling where no idle time is allowed between the successive operations of a same job, the non-idling machine constraint has just begun to receive research attention in the literature. To the best of our knowledge, the first work on such problems concerns the earliness-tardiness single-machine scheduling problem with no unforced idle time, where a Branch and Bound approach has been developed [4]. More recently, some aspects of the impact of the non-idling constraint on the complexity of single-machine scheduling problems as well as the important role played by the earliest starting time of a non-idling schedule has been studied in [2]. Moreover, in [3] and [8], exact methods have been designed to solve the basic one-machine non-idling problem and in [9], approximation algorithms have been developped for the non-idling single-machine scheduling problem with release and delivery times.

Clearly, this paper is theory oriented, and aims at a better understanding of the way non Idling constraints should be handled inside a scheduling process. When related to a specific machine, resource or infrastructure, the Non Idling feature is something rather natural, which may noted in several industrial situations. A typical one will involve an oven for pneumatic industry. As a matter of fact, Non Idle time handling is usually a cost issue, rather than a constraint. Due to safety concerns, human resource or energy requirements, it happens that costs related to stopping or starting the activity of some equipment may exceed in a drastic way other running costs, as well as costs related to due dates or penalties. Thus, it comes that non Idling constraints have to be understood as a kind of extreme case which allows us to cast the problems which arise from such a strongly non continuous cost structure into a formal framework. While in other papers devoted to Non Idling constraint management, those constraints were involving every machine or resource independently from each other, we focus here on situations which impose this constraint to any group of machines, all considered as being identical. As it will be shown a little further, it is then possible to turn the resulting constraint into a constraint on the function which, with any instant t, associates the amount of resources which are active at time t. Once again, it is a matter of cost structure. When identical machines are strongly interconnected and involve a common source of additional resources (energy, human resources), then the cost which derive from stopping or starting a given machine may be high (in comparison with other costs) and depend in a decreasing on the current number of active machine. As a matter of fact, one may notice that minimizing those costs leads to concentrating task execution into active periods with high parallelism rates, which means to schedules which induces the pyramidal shape functions which will be at the core of the next sections.

In this paper, we study the basic homogeneous m-machine non-idling problem where weakly dependent unit-time jobs have to be scheduled within their time windows so that the non-idling constraint must be satisfied not only for each machine but for every subset of machines. When the jobs are independent, an approach based on transversals [11] leads to an algorithm, whose complexity is exponential in the number of jobs, solving the problem as the search of a source-sink path in a graph whose nodes correspond to time-consecutive suitable triplets of transversals and where there is an arc linking two triplets if the two last tranversals of the first triplet correspond to the two first transversals of the second triplet. In Section 2, the problem and the key notions of pyramidal structure, k-hole, m-matching, preschedule, k-schedule and schedule are

defined. Section 3 provides feasibility criteria for the existence of an m-matching and a preschedule. Section **??** provides a necessary and sufficient condition for an instance to have at least a schedule. A conclusion finally resumes the main results of the paper and relates about future works and open questions about that basic m-machine non idling scheduling problem.

2 Problem definition

2.1 Preliminary definitions

We consider the discrete time space $\{1, 2, \cdots\}$ each element of which is called a time-unit. An *interval* of $\{1, 2, \cdots\}$ is made of a finite number of consecutive time-units. The smallest (respectively largest) time-unit of an interval Θ is denoted by $a(\Theta)$ (respectively $b(\Theta)$). If p and q are two distinct natural numbers, the interval whose bounds are p and q is denoted by $I(p, q)$ (note that we may have $p < q$ or $q < p$). Interval Θ_2 is said to dominate interval Θ_1 (what is denoted by $\Theta_1 \propto \Theta_2$) if $b(\Theta_1) + 1 < a(\Theta_2)$. If $\Theta_1 \propto \Theta_2$, we denote by $Mid(\Theta_1, \Theta_2)$ the (non-empty) interval $[b(\Theta_1) + 1, a(\Theta_2) - 1]$. By convention, two intervals Θ_1 and Θ_2 are said to be *connected* if $\Theta_1 \cup \Theta_2$ is an interval.

2.2 The homogeneous non-idling scheduling problem

We are given a set $J = \{J_1, \cdots, J_n\}$ of n unit-time jobs that are to be processed on a set $M = \{M_1, \cdots, M_m\}$ of m identical machines. Job J_i must be executed within a given time-window $F(i) = \{r_i, \cdots, d_i\}$ which is an interval. It will be convenient to denote by r_{min} (respectively d_{max}) the smallest r_i (respectively largest d_i) and by $\mathcal{H}$ the interval $[r_{min}, d_{max}]$. The jobs are constrained by a *weak* precedence relation denoted by $\preceq$ where $J_i \preceq J_j$ means that J_j must not be performed before J_i. A schedule must also satisfy the so-called *homogeneous non-idling* (HNI in short) constraint: for any subset $M' \subseteq M$, the set of the time-units at which *at least one machine* in M' does not idle away is an interval. In Figure 1, the schedule on the left does not satisfy the HNI constraint since, for the subset $\{M_1, M_3\}$, the set of the times units when M_1 or M_3 is busy is not an interval. On the contrary, the schedule on the right is an HNI-schedule.

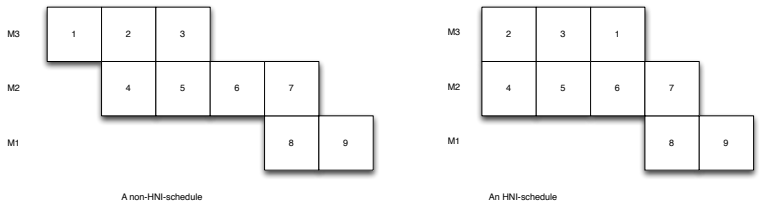


Fig. 1 A HNI-schedule and a non-HNI schedule

A schedule (T, μ) assigns a time-unit $T(i)$ and a machine $\mu(i)$ to each job J_i so that :

1. $\forall J_i \in J, T(i) \in F(i)$;
2. $\forall t \in \mathcal{H}, |\{i|T(i) = t\}| \leq m$;
3. $J_i \preceq J_j \Rightarrow T(i) \leq T(j)$;
4. for any $M' \subseteq M$, the time units t at which there is at least a job J_i processed at t (i.e: $T(i) = t$) by a machine of M' ((i.e: $\mu(i) \in M'$) make a single interval.

The problem Π_0 is to decide whether a given instance $(J, F, \preceq, m)$ has at least one schedule. If the answer is yes, the instance is said to be *feasible*. It must be pointed out that the precedence relation $\preceq$ has not the same meaning as the classical one since if $J_i \preceq J_j$, then J_i and J_j may be processed at the same time-unit. Clearly, due to the machine constraint, there is no difference when $m = 1$. It is also of interest to note that when $\preceq$ has the usual meaning, the corresponding problem is NP-complete, since in the reduction of the CLIQUE problem described in [7], the schedule of the instance corresponding to a "yes" instance of CLIQUE satisfies the HNI constraint.

Let $I = (J, F, \preceq, m)$ be an instance of Π_0. A time function T that satisfies the first two conditions will be called a *m-matching* of I. A function T that satisfies the first three conditions will be called a *preschedule* of I.

Let T be an m-matching of I. We denote by $s(T)$ (respectively $e(T)$) the smallest (respectively largest) time-unit such that at least one job is processed in T. If $s(T) \leq t \leq e(T)$, we denote by $n_T(t)$ the number of jobs scheduled at time-unit t.

(T, μ) is said to be an *histogram schedule* of Π_0 if, for any time unit t such that $n_T(t) > 0$, $\{M_1, \cdots, M_{n_T(t)}\}$ is the set of the busy machines at time unit t. Clearly, if (T, μ) is a schedule, we get an equivalent (i.e: same starting times of the tasks) histogram schedule by reassigning the tasks processed at time unit t to the machines $M_1, \cdots, M_{n_T(t)}$. Figure 2 shows a non-histogram schedule and an equivalent histogram schedule.

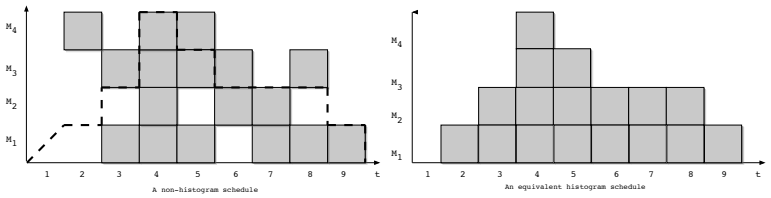


Fig. 2 A non-histogram schedule and an equivalent histogram schedule

So, from now on, we will only consider histogram schedules and we notice that an histogram schedule is completely defined by the function T. The following property characterizes the histogram schedules of an instance of Π_0 in terms of the shape of the function $n_T(t)$. The shape of $n_T(t)$ is said to be "pyramidal" if there do not exist 3 time units t_0, t_1, t_2 such that $t_0 < t_1 < t_2$, $n_T(t_0) > n_T(t_1)$, and $n_T(t_2) > n_T(t_1)$ (See Figure 3).

Property 1 Let T be a preschedule of an instance of Π_0. T is an histogram schedule if and only if the function $n_T(t)$, has a pyramidal shape.

Proof Assume that T is an histogram schedule and that the function $n_T(t)$ has not a pyramidal shape. Then there exist 3 time units t_0, t_1, t_2 such that $t_0 < t_1 < t_2$, $n_T(t_0) > n_T(t_1)$, and $n_T(t_2) > n_T(t_1)$. Clearly machine $M_{n_T(t_1)+1}$ is idle at time unit

t_1 while it is busy at time units t_0 and t_2. So T is not a schedule. Conversely, if the function $n_T(t)$ of a preschedule T has a pyramidal shape, then T clearly satisfies the HNI condition and thus is an histogram schedule.

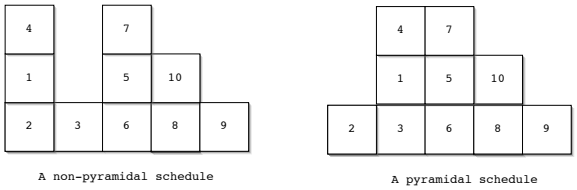


Fig. 3 A pyramidal function $n_T(t)$

The family of time windows $F(i), i \in \{1, \cdots, n\}$ is said to be *consistent* with the precedence relation $\preceq$ if:

$$J_i \preceq J_j \Rightarrow r_i \leq r_j \text{ and } d_i \leq d_j$$

The following property allows to restrict our study to instances of Π_0 such that F is consistent with respect to $\preceq$.

Property 2 Let $I = (J, F, \preceq, m)$ be an instance of Π_0. There is an instance $I' = (J, F', \preceq, m)$ of Π_0 with the same set of schedules as I such that F' is consistent with respect to $\preceq$ and such that for any job J_i, $F'(i) \subseteq F(i)$.

Proof Assume F is not consistent and let J_i be a minimal job (with respect to $\preceq$) with an immediate successor J_j that violates the consistency condition. If $r_j \leq r_i$ and $d_j \leq d_i$, then, in any schedule, J_i and J_j are scheduled in the interval $\{r_i, \cdots, d_j\}$. We thus put $F'(i) = F'(j) = \{r_i, \cdots, d_j\}$. Assume now that $r_j \geq r_i$ and $d_j \leq d_i$. Then, in any schedule, J_i is scheduled in the interval $\{r_i, \cdots, d_j\}$. We thus put $F'(i) = \{r_i, \cdots, d_j\}$. Assume finally that $r_j \leq r_i$ and $d_j \geq d_i$. Then, in any schedule, J_j is scheduled in the interval $\{r_i, \cdots, d_j\}$. We thus put $F'(j) = \{r_i, \cdots, d_j\}$. Iterating this transformation as long as the consistency condition is violated, we finally get a family F' which is consistent with respect to $\preceq$.

So, from now on, the consistency of F with respect to $\preceq$ will be assumed for any instance $(J, F, \preceq, m)$ of Π_0.

3 Existence of a preschedule

In this section, we first use the Hall's marriage theorem [10] to get a characterization of the instances for which an m-matching exists. Then we show that the existence of an m-matching implies the existence of a preschedule. If Θ is an interval, it will be useful to denote by $J(\Theta)$ the subset of the jobs J_i such that $F(i) \subseteq \Theta$.

3.1 Existence of an m-matching

The following property is a straightforward application of the Hall's marriage theorem:

Property 3 The instance $(J, F, \preceq, m)$ of Π_0 has at least one m-matching if and only if, for any interval Θ of $\mathcal{H}$, we have $|J(\Theta)| \leq |\Theta| \times m$

Proof From the Hall's marriage theorem, we know that the instance $(J, F, \preceq, m)$ of Π_0 has at least one m-matching if and only if for any subset I of J, we have $|I| \leq m \times |\Gamma(I)|$ where $\Gamma(I) = \cup_{i \in I} F(i)$. If $I = J(\Theta)$, then we have $\Gamma(I) \subseteq \Theta$ so that $|\Gamma(I)| \leq |\Theta|$ and $m \times |\Theta| \geq m \times |\Gamma(I)| \geq |I| = |J(\Theta)|$.
Conversely, let I be an arbitrary subset of J. Clearly $\Gamma(I)$ is the union $\cup_{k=1}^{r} \Theta_k$ of non-empty and pairwise disjoint intervals. With interval Θ_k is associated the subset I_k of the jobs J_i of I such that $F(i) \subseteq \Theta_k$. The subsets $I_1, \cdots, I_r$ make a partition of I such that $\Gamma(I_k) = \Theta_k$. Since $I_k \subseteq J(\Theta_k)$, we have

$$m \times |\Gamma(I)| = m \times \sum_{k=1}^{r} |\Theta_k| \geq \sum_{k=1}^{r} |J(\Theta_k)| \geq \sum_{k=1}^{r} |I_k| \geq |I|.$$

3.2 Existence of a preschedule

We now show that any m-matching of the instance I may be transformed into a preschedule.

Property 4 Let $I = (J, F, \preceq, m)$ be an instance of Π_0. I has at least one preschedule if and only if I has at least one m-matching.

Proof Let $\preceq^*$ be the transitive closure of $\preceq$ and let T be an m-matching that is not a preschedule. (J_i, J_j) is called an *inversion* of T if $T(i) > T(j)$ and $J_i \preceq^* J_j$. If (J_i, J_j) is an inversion of T, let $\Delta(i, j) = T(i) - T(j) - 1$ be its length and assume that (J_p, J_q) is an inversion of T with a minimum length.
If $\Delta(p, q) = 0$ then, by exchanging J_p and J_q in T, we do not create any new inversion and remove the inversion (J_p, J_q).
Assume now that $\Delta(p, q) > 0$ and let J_l be an arbitrary job such that $T(q) < T(l) < T(p)$. We do not have $(J_l \preceq^* J_p)$ since otherwise (J_l, J_q) is an inversion of T with a length smaller than $\Delta(p, q)$. Similarly, we do not have $(J_q \preceq^* J_l)$ since otherwise (J_p, J_l) is an inversion of T with a length smaller than $\Delta(p, q)$. So, by exchanging J_p and J_q in T, we do not create any new inversion and remove the inversion (p, q).
Finally, iterating the above transformation as long as the current m-matching has at least one inversion, we stop after at most $O(n^2)$ iterations and get a preschedule.

3.3 Propagation paths of an m-matching

Given an m-matching T, we now define the *propagation paths of* T. The *propagation graph* $G(T) = (\mathcal{H}, E(T))$ is the directed graph such that $(t, t') \in E(T)$ if $t \neq t'$ and there is at least one job J_i scheduled at t such that $t' \in F(i)$. If job J_i is scheduled at t and if $t' \in F(i)$, then J_i is said to be a *label* of the arc (t, t').

A *propagation path* of T is an elementary path $\gamma = (t_0, \cdots, t_k)$ of $G(T)$. The subpath of γ from t_i to t_j will be denoted by $\gamma(t_i, t_j)$. The length $L(\gamma)$ of γ is $k+1$ (i.e: number of vertices of γ) and the *extended length* $\hat{L}(\gamma)$ of γ is the sum $\sum_{s=1}^{k} |t_s - t_{s-1}|$. A propagation path γ is *monotone* if the sequence $(t_0, \cdots, t_k)$ is descending. A propagation path γ is *no-cross* if for any $r \in \{1, \cdots, k\}$, we have either $t_r > \max_{i \in \{1,\cdots,r-1\}}\{t_i\}$ or $t_r < \min_{i \in \{1,\cdots,r-1\}}\{t_i\}$ (See Figure 4).

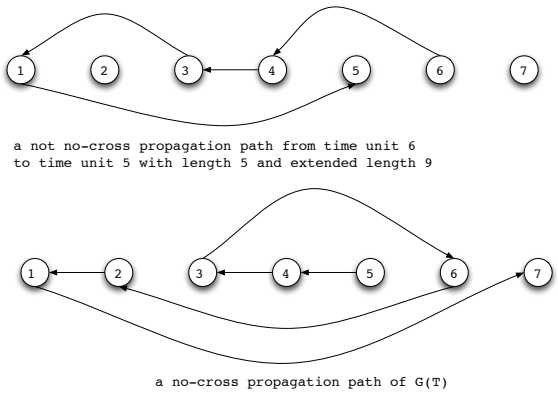


Fig. 4 Propagation paths

A propagation path is *labelled* when all its arcs are assigned a label and is said to be *fitted* if it is labelled and satisfies $n_T(t_k) < m$. It is important to note that, from the consistency property, it may be assumed that the label of an arc (t, t') is a job which is *minimal with respect to* $\preceq$ among the jobs scheduled at time t by T. Only such labelled propagation paths will be considered in the rest of the paper.

If $\gamma = (t_0, \cdots, t_k)$ is a propagation path of $G(T)$ fitted with the labelling $\sigma = (J_{[0]}, \cdots, J_{[k-1]})$, we get an other m-matching $T' = \mathcal{T}(T, \gamma, \sigma)$ by putting $T'(J_{[p]}) = t_{p+1}$ for all $p \in \{0, \cdots, k-1\}$. Figure 5 illustrates this transformation. Let T be a preschedule and let $\gamma = (t_0, \cdots, t_k)$ be a propagation path of $G(T)$. If γ has at least a labelling σ such that $T' = \mathcal{T}(T, \gamma, \sigma)$ is a preschedule, then γ is said to be compatible with $\preceq$ ($\preceq$-*compatible* in short).

The aim of the two following lemmas is to show that no-cross propagation paths allow to move from a preschedule into another one.

Property 5 Let T be a preschedule and assume that $\gamma = (t_0, \cdots, t_q)$ is a propagation path of $G(T)$ from $t_0 = u$ to $t_q = v$. Then there also exists a no-cross propagation path from u to v in $G(T)$.

Proof Assume that γ is not no-cross and let t_r $(2 \leq r \leq q)$ be the first node of γ such that $\min_{i \in \{1,\cdots,r-1\}}\{t_i\} < t_r < \max_{i \in \{1,\cdots,r-1\}}\{t_i\}$. From the definition of t_r, we know there is a smallest index s $(0 \leq s \leq r-2)$ such that t_r belongs to $I(t_s, t_{s+1})$. Thus $(t_s, t_r) \in E(T)$ and $\gamma(t_0, t_s) \cdot (t_s, t_r)$ is no-cross. The above transformation may then be iterated while the current propagation path is not no-cross.

The next lemma shows we do not need take care of the $\preceq$-compatibility since if there is a fitted propagation path from time unit t_0 to time unit t_q, there also exists a $\preceq$-compatible fitted propagation path from t_0 to t_q .

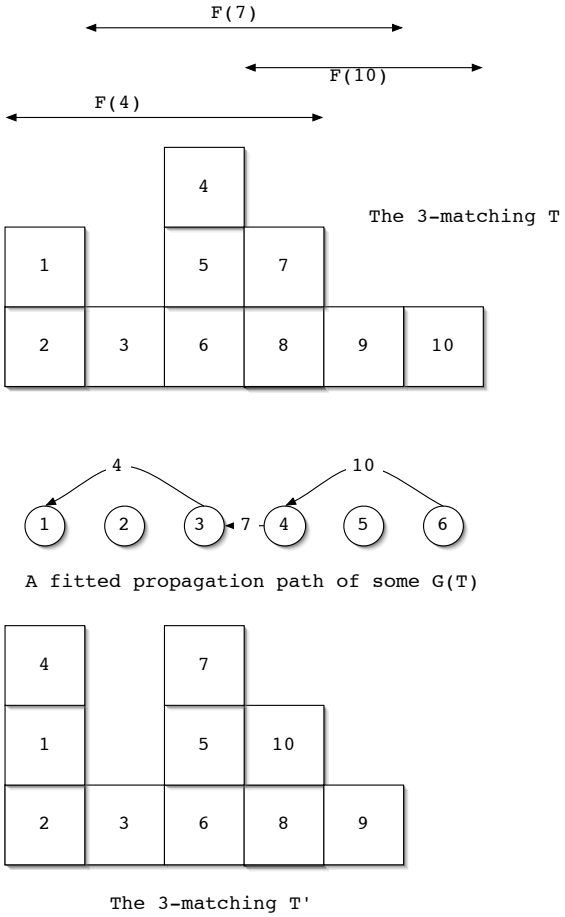


Fig. 5 The new m-matching $T' = \mathcal{T}(T, \gamma, \sigma)$

Property 6 Let T be a preschedule and assume that $(t_0, \cdots, t_q)$ is a propagation path of $G(T)$ from $t_0 = u$ to $t_q = v$ (where $n_T(t_q) < m$). Then there exists a no-cross and $\preceq$-compatible propagation path from u to v in $G(T)$.

Proof Let $\gamma = (t_0, \cdots, t_q)$ be a no-cross propagation path from u to v whose extended length $\hat{L}(\gamma)$ is minimum and whose length $L(\gamma)$ is maximum among the propagation paths with a minimum extended length. Assume also that γ is not compatible with $\preceq$ and let $\sigma = (J_{[0]}, \cdots, J_{[q-1]})$ be a labelling of γ. Jobs $J_{[0]}, \cdots, J_{[q-1]}$ are called the *moving jobs* while the other jobs are the *static jobs*. Since T is a preschedule and γ is not $\preceq$-compatible, $\preceq$ is violated in $T' = \mathcal{T}(T, \gamma, \sigma)$ either because an inversion has been created between a static job J' scheduled at t' and a moving job $J_{[s]}$ scheduled at t_{s+1} (case (1) below) or between two moving jobs $J_{[r]}$ and $J_{[s]}$ respectively scheduled at time t_{r+1} and t_{s+1} (case (2) below).

Case (1).

For that case, we either have ($J' \preceq J_{[s]}$ and $t_{s+1} < t'$) or ($J_{[s]} \preceq J'$ and $t_{s+1} > t'$). Since these two subcases are symmetric and the same line of reasoning applies to both, we will only consider the first one.

Let us assume that t' is not a node of γ. Then consider the smallest index p such that $t' \in I(t_p, t_{p+1})$. Assume that $p = s$. Since $J' \preceq J_{[s]}$, we have $(t', t_{s+1}) \in E(T)$ from the consistency of F. Since $t_{s+1} < t' < t_s$, we have $(t_s, t') \in E(T)$. Now $\gamma' = \gamma(t_0, t_s) \cdot (t_s, t') \cdot (t', t_{s+1}) \cdot \gamma(t_{s+1}, t_q)$ is a no-cross propagation path from $t_0 = u$ to $t_q = v$ such that $\hat{L}(\gamma') = \hat{L}(\gamma)$ and $L(\gamma') > L(\gamma)$. A contradiction.

Consider now the case when $p < s$. From the definition of p, we know that $(t_p, t') \in E(T)$. Moreover, since $J' \preceq J_{[s]}$, we have $(t', t_{s+1}) \in E(T)$ from the consistency of F (see Figure 6). Since γ has a minimum extended length and from the definition of p, we have that $\gamma(t_0, t_p) \cdot (t_p, t') \cdot (t', t_{s+1}) \cdot \gamma(t_{s+1}, t_q)$ is a no-cross propagation path whose extended length is strictly smaller than $\hat{L}(\gamma)$, what is a contradiction.

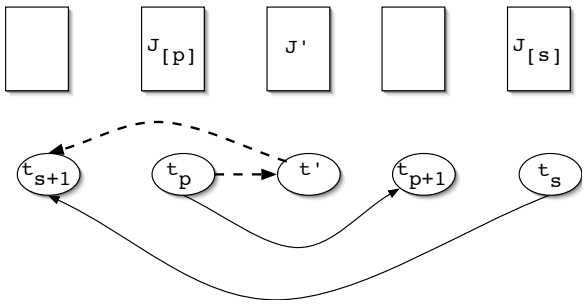


Fig. 6 $J' \preceq J_{[s]}$, $t_{s+1} < t'$ and $t' \notin \gamma$

Assume now that t' is a node of γ and more precisely that $t' = t_p$ where $0 \leq p \leq q-1$ (see Figure 7).

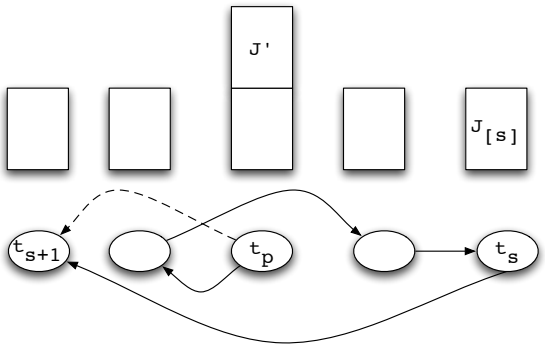


Fig. 7 $J' \preceq J_{[s]}$, $t_{s+1} < t'$ and $t' \in \gamma$

Since γ is no-cross, we have $p \leq s$. Morever, since $J_{[s]}$ is a minimal job (with respect to $\preceq$) among the jobs scheduled at time unit t_s, we cannot have $J' \preceq J_{[s]}$. So we know that

$p < s$. Since $J' \preceq J_{[s]}$, we have $(t', t_{s+1}) \in E(T)$. Now $\gamma(t_0, t_p) \cdot (t_p, t_{s+1}) \cdot \gamma(t_{s+1}, t_q)$ is a no-cross propagation path whose extended length is strictly smaller than $\hat{L}(\gamma)$, what is a contradiction.

Case (2).

As shown in Figure 8, let us assume that $J_{[r]} \preceq J_{[s]}$ and $t_{s+1} < t_{r+1}$ (the symmetric case may be treated in a same way).

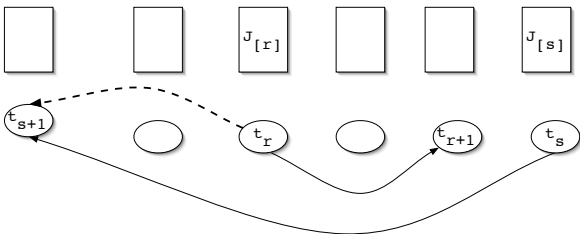


Fig. 8 $J_{[r]} \preceq J_{[s]}$ and $t_{s+1} < t_{r+1}$

Since γ is no-cross, we must have $r < s$. Since $J_{[r]} \preceq J_{[s]}$, we have $(t_r, t_{s+1}) \in E(T)$ from the consistency of F. So, $\gamma(t_0, t_r) \cdot (t_r, t_{s+1}) \cdot \gamma(t_{s+1}, t_q)$ is a no-cross propagation path whose extended length is strictly smaller than $\hat{L}(\gamma)$, what is a contradiction.
We thus may conclude that γ is a no-cross and $\preceq$-compatible propagation path from u to v in $G(T)$.

Let T be a preschedule and let u be a time unit such that $n_T(u) > 0$. The next lemma gives an important property of the set $A_T(u)$ of the time units that may be reached by the propagation paths of $G(T)$.

Property 7 $A_T(u)$ is a T-block.

Proof Assume first that $A_T(u)$ is made of more than one interval. Since $u \in A_T(u)$, we may denote by Θ_0 the interval of $A_T(u)$ that contains u. Let Θ_1 be an interval of $A_T(u)$ that is a neighbour of Θ_0 and assume for example that $\Theta_0 \propto \Theta_1$. Let $t \in Mid(\Theta_0, \Theta_1)$ and $v \in \Theta_1$. There is a propagation path $\gamma = (t_0, \cdots, t_k)$ from u to v in $G(T)$. Let (t_q, t_{q+1}) be the first arc of γ such that $t_q \leq b(\Theta_0)$ and $t_{q+1} \geq a(\Theta_1)$. Then, for any job J_i that is a label of (t_q, t_{q+1}), we have that $t \in F(i)$, so that we may conclude that $t \in A_T(u)$, a contradiction. Thus $A_T(u)$ is an interval. Assume now that $T(i) = t \in A_T(u)$. If $t' \in F(i) \setminus A_T(u)$, then $(t, t') \in E(T)$ and we would have $t' \in A_T(u)$, a contradiction. $A_T(u)$ is thus a T-block.

The two following properties, which concern the structure of T-blocks will be quite useful for the proof the main theorem.

Property 8 Let T be an m-matching and let Θ_1,Θ_2 be two connected T-blocks. Then $\Theta_1 \cup \Theta_2$ is a T-block and $\lambda(\Theta_1 \cup \Theta_2) \geq \max\{\lambda(\Theta_1), \lambda(\Theta_2)\}$.

Proof If job J_i is scheduled in Θ_1, then $F(i) \in Int(\Theta_1) \subseteq Int(\Theta_1 \cup \Theta_2)$ since Θ_1 is a T-block. In the same way, if job J_i is scheduled in Θ_2, then $F(i) \in Int(\Theta_2) \subseteq Int(\Theta_1 \cup \Theta_2)$. So $\Theta_1 \cup \Theta_2$ is a T-block. Assume that $\lambda(\Theta_1) = \lceil \frac{|J(\Theta'_1)|}{|\Theta'_1|} \rceil$ where $\Theta'_1 \in Int(\Theta_1)$. Since $\Theta'_1 \in Int(\Theta_1 \cup \Theta_2))$, we get $\lambda(\Theta_1) \leq \lambda(\Theta_1 \cup \Theta_2)$. In the same way, we get $\lambda(\Theta_2) \leq \lambda(\Theta_1 \cup \Theta_2)$. We thus conclude that $\lambda(\Theta_1 \cup \Theta_2) \geq \max\{\lambda(\Theta_1), \lambda(\Theta_2)\}$.

Property 9 Let T be an m-matching, let Θ_1 be a T-block and let Θ_2 be an interval such that $\Theta_1 \cap \Theta_2 = \emptyset$. If for any $(u,t) \in \Theta_1 \times \Theta_2, n_T(u) \geq n_T(t)$. If, in addition, there is $t \in \Theta_1$ such that $n_T(t) > \max_{u \in \Theta_2} n_T(u)$, then $\lambda(\Theta_1) > \lambda(\Theta_2)$.

Proof Let Θ be an arbitrary interval of Θ_2 (i.e: $\Theta \in Int(\Theta_2)$). Every job of $J(\Theta)$ is scheduled in Θ so that we have $|J(\Theta)| \leq \sum_{t \in \Theta} n_T(t)$. Since Θ_1 is a T-block, $J(\Theta_1)$ is exactly the set of the jobs scheduled in Θ_1 so that we have:

$$\lambda(\Theta_1) \geq \lceil \frac{|J(\Theta_1)|}{|\Theta_1|} \rceil = \lceil \frac{\sum_{u \in \Theta_1} n_T(u)}{|\Theta_1|} \rceil$$

Assume that $M = \max_{t \in \Theta} n_T(t)$. Then we have $|J(\Theta)| \leq M|\Theta|$ and $|J(\Theta_1)| = \sum_{u \in \Theta_1} n_T(u) \geq M|\Theta_1|$. We thus derive that

$$\lambda(\Theta_1) \geq \lceil \frac{|J(\Theta_1)|}{|\Theta_1|} \rceil \geq M \geq \lceil \frac{|J(\Theta)|}{|\Theta|} \rceil$$

and we may conclude that $\lambda(\Theta_1) \geq \lambda(\Theta_2)$.

For the special case when there is $t \in \Theta_1$ such that $n_T(t) > M$, we have

$$\lambda(\Theta_1) \geq \lceil \frac{\sum_{u \in \Theta_1} n_T(u)}{|\Theta_1|} \rceil > M$$

and we may conclude that $\lambda(\Theta_1) > \lambda(\Theta_2)$.

3.4 Feasibility of an instance

We are now able to provide a characterization of the instances $(J, F, \preceq, m)$ which are feasible for the problem Π_0.

Theorem 1 *The instance $(J, F, \preceq, m)$ of Π_0 is feasible if and only if the following two conditions are satisfied :*

1. *for any interval Θ of $\mathcal{H}$, $|J(\Theta)| \leq |\Theta| \times m$;*
2. *for any sequence $(\Theta_1, \cdots, \Theta_p)$ of intervals of $\mathcal{H}$ such that $\Theta_1 \propto \cdots \propto \Theta_p$, we have*

$$|J \setminus \cup_{s=1}^{p} J(\Theta_s)| \geq \sum_{s=1}^{p-1} \mu(\Theta_s, \Theta_{s+1})$$

The "only if" part of the proof is a straightforward consequence of Properties 3. In order to prove the "if" part, we introduce the analog of Property 1 for k-schedules and prove it by induction on k.
Recall that, for $k \in \{1, \cdots, m\}$, a preschedule T is a *k-schedule* if T has no l-hole for any $l \in \{0, \cdots, k-1\}$.

We now adapt the definition of $\mu(\Theta_1, \Theta_2)$ (where Θ_1 and Θ_2 are two time intervals such that $\Theta_1 \propto \Theta_2$) to k-schedules by defining $\tilde{\mu}(\Theta_1, \Theta_2, k)$ by the number $|Mid(\Theta_1, \Theta_2)| \times \min\{\lambda(\Theta_1), \lambda(\Theta_2), k\}$. Then have the following property:

Property 10 Let Θ_1 and Θ_2 be two time intervals such that $\Theta_1 \propto \Theta_2$. In any k-schedule of $(J, F, \preceq, m)$, at least $\tilde{\mu}(\Theta_1, \Theta_2, k)$ jobs are scheduled in the interval $Mid(\Theta_1, \Theta_2)$.

We now prove the analog of Property 1 for k-schedules.

Property 11 The instance $(J, F, \preceq, m)$ of Π_0 has at least one k-schedule if and only if the following two conditions are satisfied :

1. for any Θ of $\mathcal{H}$, $|J(\Theta)| \leq |\Theta| \times m$;
2. for any sequence $(\Theta_1, \cdots, \Theta_p)$ of intervals of $\mathcal{H}$ such that $\Theta_1 \propto \cdots \propto \Theta_p$, we have

$$|J \setminus \cup_{s=1}^{p} J(\Theta_s)| \geq \sum_{s=1}^{p-1} \tilde{\mu}(\Theta_s, \Theta_{s+1}, k)$$

Proof Let $I = (J, F, \preceq, m)$ be an instance of Π_0. The "only if" part of the proof comes directly from Properties 3 and 10. For the "if" part, we assume that the property is true for k-schedules and we assume that the instance I has no $(k+1)$-schedule. If I has at least a preschedule but has no k-schedule, then we know from the induction that there is a sequence $(\Theta_1, \cdots, \Theta_p)$ of time intervals such that $\Theta_1 \propto \cdots \propto \Theta_p$ and

$$|J \setminus \cup_{s=1}^{p} J(\Theta_s)| < \sum_{s=1}^{p-1} \tilde{\mu}(\Theta_s, \Theta_{s+1}, k)$$

Since $\tilde{\mu}(\Theta_s, \Theta_{s+1}, k) \leq \tilde{\mu}(\Theta_s, \Theta_{s+1}, k+1)$, we also have

$$|J \setminus \cup_{s=1}^{p} J(\Theta_s)| < \sum_{s=1}^{p-1} \tilde{\mu}(\Theta_s, \Theta_{s+1}, k+1)$$

and we are done.
We now assume that I has at least one k-schedule and that any such schedule has at least a k-hole. Let $\Sigma_k(I)$ be the set of the k-schedules of I. If $T \in \Sigma_k(I)$, we need some notations that are attached to T but for which the reference to T will be omitted to get simpler notations:

- H is the set of k-holes and $h = |H|$;
- $S_p, p \in \{1, \cdots, m\}$ is the set of the time units t such that $n_T(t) = p$ and $N_p = |S_p|$;
- $\hat{S} = \cup_{p=k+2}^{m} S_p$;
- u^- (respectively u^+) is the smallest (respectively largest) k-hole;
- for $p \in \{1, \cdots, k+1\}$, C_p^- (respectively C_p^+) is the smallest (respectively largest) time unit t such that $n_T(t) \geq p$;

Note that for $p \in \{1, \cdots, k+1\}$, we have $C_p^- < u^-$ and $C_p^+ > u^+$ and that for $p \in \{2, \cdots, k+1\}$, we have $C_{p-1}^- \leq C_p^-$ and $C_{p-1}^+ \geq C_p^+$. Figure 9 illustrates these notations.

Since $\Sigma_k(I)$ is not empty, we consider one of its schedules, denoted by T, such that:

1. h is minimum;
2. the vector $(N_m, \cdots, N_1)$ is minimal for the lexicographic order among the schedules such that h is minimum.

From Property 6 and Condition 1., we know that there is no propagation path in $G(T)$ from a time unit of $\hat{S}$ to a time unit of H. Similarly, there is no propagation path γ in $G(T)$ from a time unit $t \in \hat{S}$ to a time unit $C_p^+ + 1$ (or $C_p^- - 1$) for $p \in \{1, \cdots, k+1\}$ since otherwise a labelling σ of γ would exist such that $T' = \mathcal{T}(T, \gamma, \sigma)$ would be a

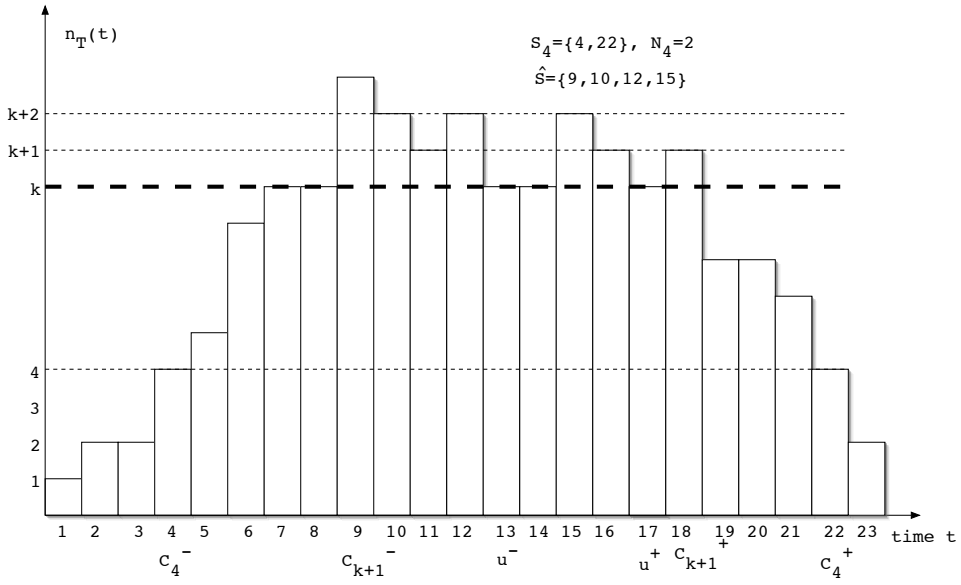


Fig. 9 A schedule T of $\Sigma_k(I)$

k-schedule such that, if $t \in S_l$ ($l \geq k+2$), we would have $N'_l = N_l - 1$ and $N'_q = N_q$ for $q \in \{k+2, \cdots, m\} \setminus \{l\}$. So T would not satisfy Condition 2.

For $u \in \hat{S}$, it comes from Property 7 that $A_T(u)$ is a T-block satisfying $A_T(u) \subseteq S_{k+1} \cup \hat{S}$ and $\lambda(A_T(u)) \geq k+2$. Let us now consider the connected components of the intersection graph of the family of intervals $A_T(u)$ for $u \in \hat{S}$. We know from Property 8 that the intervals associated with these components are pairwise disjoint T-blocks that we denote by $\mathcal{M}_1, \cdots, \mathcal{M}_s$ and for which we may assume without loss of generality that $\mathcal{M}_1 \propto \cdots \propto \mathcal{M}_s$. From the definition of $\mathcal{M}_1, \cdots, \mathcal{M}_s$, we have:

$$\hat{S} \subseteq \mathcal{M}_1 \cup \cdots \cup \mathcal{M}_s \subseteq S_{k+1} \cup \hat{S}$$

and from Property 8, we also have $\lambda(\mathcal{M}_i) \geq k+2$ for $i \in \{1, \cdots, s\}$. Now let $i \in \{1, \cdots, s-1\}$. Any $t \in Mid(\mathcal{M}_i, \mathcal{M}_{i+1})$ is such that $n_T(t) \leq k+1$ so that we have:

$$\sum_{t \in Mid(\mathcal{M}_i, \mathcal{M}_{i+1})} n_T(t) \leq (k+1)|Mid(\mathcal{M}_i, \mathcal{M}_{i+1})| = \tilde{\mu}(\mathcal{M}_i, \mathcal{M}_{i+1}, k+1)$$

Consider now time unit $t = C_p^+$ for $p \in \{1, \cdots, k+1\}$. Again from Condition 1., there is no propagation path in $G(T)$ from t to a time-unit of H and, from Condition 2., there is no propagation path in $G(T)$ from t to time unit $C_{p-1}^+ + 1$. So, the intervals $A_T(C_p^+)$ are T-blocks such that $\lambda(A_T(C_p^+)) \geq p$ and are all included in $(S_{p-1} \cup S_p \cup \cdots \cup S_{k+1} \cup \hat{S}) \setminus H$. Considering the connected components of the intersection graph of the intervals $A_T(C_p^+)$ for $p \in \{1, \cdots, k+1\}$, we get a family $\mathcal{R}_1, \cdots, \mathcal{R}_r$ of T-blocks such that:

1. $\mathcal{R}_r \propto \cdots \propto \mathcal{R}_1$;
2. $\lambda(\mathcal{R}_r) \geq k+1$ (since $A_T(C_{k+1}^+) \subset \mathcal{R}_r$ and Property 8);
3. $\lambda(\mathcal{R}_r) > \cdots > \lambda(\mathcal{R}_1)$ (Property 9);

4. for any $p \in \{1, \cdots, k+1\}$, C_p^+ belongs to an interval $\mathcal{R}_{i(p)}$ such that $\lambda(\mathcal{R}_{i(p)}) \geq p$;
5. for any $i \in \{2, \cdots, r\}$, we have:

$$\sum_{t \in Mid(\mathcal{R}_i, \mathcal{R}_{i-1})} n_T(t) = \lambda(\mathcal{R}_{i-1})|Mid(\mathcal{R}_i, \mathcal{R}_{i-1})| \leq \tilde{\mu}(\mathcal{R}_i, \mathcal{R}_{i-1}, k+1)$$

since $\lambda(R_{i-1}) = \lambda(A_T(C_p^+))$ where p is the largest index such that $C_p^+ \in \mathcal{R}_{i-1}$;
6. $u^+ < a(\mathcal{R}_r)$.

Using symmetrical arguments, we get from the time units C_p^- for $p \in \{1, \cdots, k+1\}$ a family of T-blocks $\mathcal{L}_1, \cdots, \mathcal{L}_l$ with the following properties:

1. $\mathcal{L}_1 \propto \cdots \propto \mathcal{L}_l$;
2. $\lambda(\mathcal{L}_l) \geq k+1$ (since $A_T(C_{k+1}^-) \subset \mathcal{L}_l$ and Property 8);
3. $\lambda(\mathcal{L}_1) < \cdots < \lambda(\mathcal{L}_l)$ (Property 9);
4. for any $p \in \{1, \cdots, k+1\}$, C_p^- belongs to an interval $\mathcal{L}_{j(p)}$ such that $\lambda(\mathcal{L}_{j(p)}) \geq p$;
5. for any $i \in \{1, \cdots, l-1\}$, we have:

$$\sum_{t \in Mid(\mathcal{L}_i, \mathcal{L}_{i+1})} n_T(t) = \lambda(\mathcal{L}_i)|Mid(\mathcal{L}_i, \mathcal{L}_{i+1})| \leq \tilde{\mu}(\mathcal{L}_i, \mathcal{L}_{i+1}, k+1)$$

6. $u^- > b(\mathcal{L}_l)$.

It comes from the definition of the T-block families $\mathcal{L}$, $\mathcal{R}$ and $\mathcal{M}$ that:

1. for $i \in \{1, \cdots, r-1\}$, and $j \in \{1, \cdots, s\}$, we have $\mathcal{R}_i \cap \mathcal{M}_j = \emptyset$;
2. for $i \in \{1, \cdots, l-1\}$, and $j \in \{1, \cdots, s\}$, we have $\mathcal{L}_i \cap \mathcal{M}_j = \emptyset$.

Figure 10 shows a possible configuration of these T-block families.

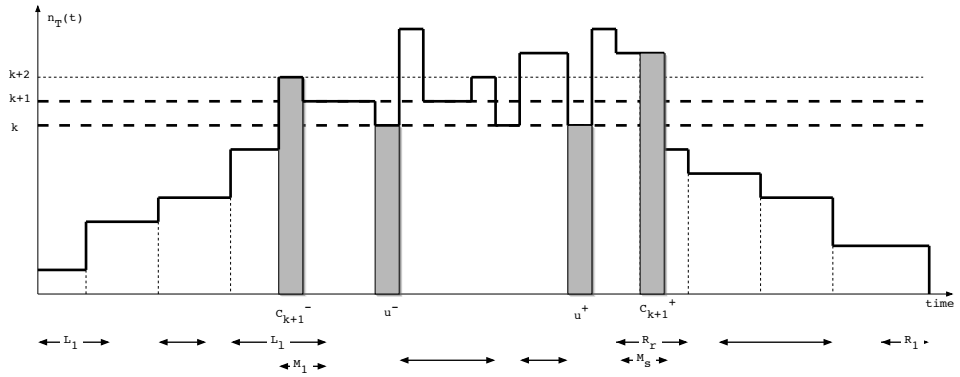


Fig. 10 The T-blocks families $\mathcal{L}$, $\mathcal{R}$,$\mathcal{M}$

Let us consider the intersection graph of the T-blocks of the 3 families $\mathcal{L}$, $\mathcal{M}$ and $\mathcal{R}$ and let us denote by $I_1, \cdots, I_q$ the T-blocks associated with its connected components. These T-blocks may be indexed so that $\mathcal{I}_1 \propto \cdots \propto \mathcal{I}_q$ and it comes from the definition of $\mathcal{L}$, $\mathcal{M}$ and $\mathcal{R}$ that:

1. for any $i \in \{1, \cdots, q\}$, $\mathcal{I}_i \cap H = \emptyset$;

2. the time units of $\hat{S}$ as well as the times units C_p^+ and C_p^- (for any $p \in \{1, \cdots, k+1\}$) belong to $\cup_{i \in \{1,\cdots,q\}} \mathcal{I}_i$;
3. any $\mathcal{I}_i$ such that $u^- < a(\mathcal{I}_i) \leq b(\mathcal{I}_i) < u^+$ is one T-block of the $\mathcal{M}$ family.

Let us assume that $C_{k+1}^- \in \mathcal{I}_{\hat{l}}$ and $C_{k+1}^+ \in \mathcal{I}_{\hat{r}}$ ($1 \leq \hat{l} < \hat{r} \leq q$). Again from the definition of the families $\mathcal{L}$, $\mathcal{M}$ and $\mathcal{R}$, we know that:

1. for $i \in \{1, \cdots, \hat{l}-1\}$, $\mathcal{I}_i$ belongs to the $\mathcal{L}$ family;
2. for $i \in \{\hat{r}+1, \cdots, q\}$, $\mathcal{I}_i$ belongs to the $\mathcal{R}$ family;
3. for $i \in \{\hat{l}, \cdots, \hat{r}\}$, we have $\lambda(\mathcal{I}_i) \geq k+1$ since
   - $A_T(C_{k+1}^-) \subseteq \mathcal{I}_{\hat{l}}$ and $\lambda(A_T(C_{k+1}^-)) \geq k+1$
   - $A_T(C_{k+1}^+) \subseteq \mathcal{I}_{\hat{r}}$ and $\lambda(A_T(C_{k+1}^+)) \geq k+1$
   - for $i \in \{\hat{l}+1, \cdots, \hat{r}-1\}$, $\mathcal{I}_i$ is a T-block of the $\mathcal{M}$ family;
4. $\lambda(\mathcal{I}_{\hat{r}}) > \lambda(\mathcal{I}_{\hat{r}+1}) > \cdots > \lambda(\mathcal{I}_q)$ (from Property 9);
5. $\lambda(\mathcal{I}_1) < \lambda(\mathcal{I}_2) < \cdots < \lambda(\mathcal{I}_{\hat{l}})$ (from Property 9).

Let $i \in \{\hat{r}+1, \cdots, q\}$ and let $p^+(i)$ be the largest number $p \in \{1, \cdots, k\}$ such that $C_p^+ \in \mathcal{I}_i$. Then we derive from Property 8 that $\lambda(\mathcal{I}_i) = \lambda(A_T(C_{p^+(i)}^+))$ and that for any $t \in Mid(\mathcal{I}_{i-1}, \mathcal{I}_i)$, we have $n_T(t) = \lambda(\mathcal{I}_i)$. Similarly, for $i \in \{1, \cdots, \hat{l}-1\}$, if $p^-(i)$ is the largest number $p \in \{1, \cdots, k\}$ such that $C_p^- \in \mathcal{I}_i$, we derive that, $\lambda(\mathcal{I}_i) = \lambda(A_T(C_{p^-(i)}^-))$ and that for any $t \in Mid(\mathcal{I}_i, \mathcal{I}_{i+1})$, we have $n_T(t) = \lambda(\mathcal{I}_i)$. Thus the following inequalities are satisfied:

1. for any $i \in \{\hat{r}, \cdots, q-1\}$,
$$\sum_{t \in Mid(\mathcal{I}_i, \mathcal{I}_{i+1})} n_T(t) = |Mid(\mathcal{I}_i, \mathcal{I}_{i+1})| \times \lambda(\mathcal{I}_{i+1}) \leq \tilde{\mu}(\mathcal{I}_i, \mathcal{I}_{i+1}, k+1)$$
2. for any $i \in \{1, \cdots, \hat{l}-1\}$,
$$\sum_{t \in Mid(\mathcal{I}_i, \mathcal{I}_{i+1})} n_T(t) = |Mid(\mathcal{I}_i, \mathcal{I}_{i+1})| \times \lambda(\mathcal{I}_i) \leq \tilde{\mu}(\mathcal{I}_i, \mathcal{I}_{i+1}, k+1)$$

Moreover, we know that for $i \in \{\hat{l}, \cdots, \hat{r}-1\}$,

$$\sum_{t \in Mid(\mathcal{I}_i, \mathcal{I}_{i+1})} n_T(t) \leq (k+1) \times |Mid(\mathcal{I}_i, \mathcal{I}_{i+1})| \leq \tilde{\mu}(\mathcal{I}_i, \mathcal{I}_{i+1}, k+1)$$

Finally, let i^+ be such that $u^+ \in Mid(\mathcal{I}_{i^+}, \mathcal{I}_{i^+ +1})$. We have $n_T(u^+) = k$ and for any $t \in Mid(\mathcal{I}_{i^+}, \mathcal{I}_{i^+ +1})$, $n_T(t) \leq k+1$. So we derive that

$$\sum_{t \in Mid(\mathcal{I}_{i^+}, \mathcal{I}_{i^+ +1})} n_T(t) < (k+1)|Mid(\mathcal{I}_{i^+}, \mathcal{I}_{i^+ +1})| \leq \tilde{\mu}(\mathcal{I}_{i^+}, \mathcal{I}_{i^+ +1}, k+1).$$

Since $I_1, \cdots, I_q$ are T-blocks, we have

$$\sum_{i=1}^{q-1} \sum_{t \in Mid(\mathcal{I}_i, \mathcal{I}_{i+1})} n_T(t) = |J \setminus (\cup_{i=1}^q J(\mathcal{I}_i))| < \sum_{i=1}^{q-1} \tilde{\mu}(\mathcal{I}_i, \mathcal{I}_{i+1}, k+1).$$

The sequence $\mathcal{I}_1, \cdots, \mathcal{I}_q$ thus matches the required conditions to prove the "if" part of the theorem.

Acknowledgements Put any acknowledgements here, or delete this section

References

1. P. Baptiste. Scheduling Unit Tasks to Minimize the Number of Idle Periods: a Polynomial Time Algorithm for Off-line Dynamic Power Management, *Research Report*, Laboratoire d'Informatique CNRS LIX, (2005).
2. P. Chrétienne, On single-machine scheduling without intermediate delays, *Discrete Applied Mathematics*, 13, 156, 2543-2550, (2008).
3. A. Jouglet, Single-machine scheduling with no idle time and release dates to minimize a regular criterion, *Journal of Scheduling*, to appear.
4. J.M.S. Valente and R.A.F.S. Alves. An Exact Approach to Early/Tardy Scheduling with Releases Dates. *Computers and Operations Research*,32, 2905-2917, (2005).
5. K. Landis. Group Technology and Cellular Manufacturing in the Westvaco Los Angeles VH Department. *Project Report in IOM 581, School of Business, University of Southern california*, (1993).
6. S. Irani and K. Pruhs. Algorithmic Problems in Power Management, *ACM Press*, vol 36, 63-76, New York, USA, (2005).
7. J. K. Lenstra and A. H. G. Rinnoy Kan. Complexity of Scheduling under Precedence Constraints, *Oper. Res.*, 26(1), 22-35, (1978).
8. J. Carlier, A. Moukrim and K. Guédira. Exact resolution of the ne-machine sequencing problem with no machine idle time, *Computers and Industrial Engineering*, 59(2), 193-199, (2010).
9. I. Kacem and H. Kellerer. Approximation algorithms for no idle time scheduling on a single machine with release times and delivery times, *Discrete Applied Mathematics*, Article in Press, (2011).
10. P. Hall. On representatives of subsets, *J. London Math. Soc.*, 10(1), 26-30, (1935).
11. M. Middendorf and V.G. Timkovsky. Transversal graphs for partially ordered sets: sequencing, merging and scheduling problems *J. Combinatorial Optimization.*, 3(4), 417-435, (1999).

MISTA 2013

Scheduling in Wireless OFDMA-TDMA Networks using Variable Neighborhood Search Metaheuristic

Pablo Adasme · Abdel Lisser · Chen Wang · Ismael Soto

Abstract In this paper, we present a hybrid resource allocation model for OFDMA-TDMA wireless networks and an algorithmic framework using a Variable Neighborhood Search metaheuristic approach for solving the problem. The model is aimed at maximizing the total bandwidth channel capacity of an uplink OFDMA-TDMA network subject to user power and subcarrier assignment constraints while simultaneously scheduling users in time. As such, the model is best suited for non-real time applications where subchannel multiuser diversity can be further exploited simultaneously in frequency and in time domains. The VNS approach is constructed upon a key aspect of the proposed model, namely its decomposition structure. Our numerical results show tight bounds for the proposed algorithm, e.g. less than 2% in most of the instances. Finally, the bounds are obtained at a very low computational cost.

1 Introduction

Orthogonal frequency and time division multiple access (resp. OFDMA, TDMA) are two wireless multi-carrier transmission schemes currently embedded into modern technologies such as Wifi and Wimax [6]. In an OFDMA network, multiple access is achieved by assigning different subsets of subcarriers (subchannels) to different users while maintaining orthogonal frequencies among subcarriers. In theory, this means that interference among subcarriers is completely minimized which allows simultaneous data rate transmissions from/to several users to/from the base station (BS). The transmission direction from the BS to users is known as a downlink process while the opposite is known as an uplink process. The TDMA transmission scheme, on the other hand, has the property of scheduling users in time by assigning all bandwidth channel capacity to only one user within a given time slot in order to transmit signals. Although, these

Pablo Adasme · Ismael Soto
Departamento de Ingeniería Eléctrica, Universidad de Santiago de Chile, Avenida Ecuador 3519 Santiago, Chile
E-mail: {pablo.adasme,ismael.soto}@usach.cl

Chen Wang · Abdel Lisser
Laboratoire de Recherche en Informatique, Université Paris-Sud XI, Bâtiment 650, 91405, Orsay Cedex France
E-mail: {chen.wang,abdel.lisser}@lri.fr

transmission schemes work differently, the underlying purpose in both of them is nearly the same, i.e. to make an efficient use of resource allocation of power and bandwidth channel capacity of the network.

In this paper, we propose a hybrid resource allocation model for OFDMA-TDMA wireless networks and an algorithmic framework using a variable neighborhood search metaheuristic approach (VNS for short) for solving the problem [3]. More precisely, we aim at maximizing the total bandwidth channel capacity of an uplink OFDMA-TDMA network subject to user power and subcarrier assignment constraints while simultaneously scheduling users in time. As such, the model is best suited for non-real time applications where signals can be transmitted at different time slots without further restrictions [4]. The latter allows the fact that subchannel multiuser diversity can be further exploited simultaneously in frequency and in time domains. As far as we know, joint OFDMA-TDMA transmission schemes have not been investigated so far. In [8], the authors compare the performance in support of real time multimedia transmission schemes when using separately OFDMA-TDMA and OFDMA networks. Their numerical results show that OFDMA outperforms OFDMA-TDMA in several quality of service metrics for real-time applications. In a similar vein, the authors in [9] consider resource allocation of an OFDM wireless network while mixing real-time and non-realtime traffic patterns. They use a utility based framework to balance efficiency and fairness among users. Thus, they propose a scheduler mechanism which gives in one shot the subcarrier and power allocation plus the transmission scheduling for each time slot. Their numerical results indicate that the proposed method achieves a significant performance in terms of the overall throughput of the system. Another related work is proposed in [10] where an hybrid transmission scheme for non-realtime applications while using simultaneously code division and time division multiple access (CDMA-TDMA) schemes is investigated. The authors use a utility based approach as well, and formulate the optimal downlink resource allocation problem for a non-realtime CDMA-TDMA network. Their numerical results show a significant improvement in the overall throughput of the system due to multi-access-point diversity gain.

We propose a simple VNS based metaheuristic approach [3] to compute tight bounds for our hybrid OFDMA-TDMA optimization problem. To this purpose, we randomly partition the set of users into T disjoint subsets of users within each iteration of the VNS approach. By doing so, we must solve T smaller integer linear programming (ILP) subproblems, one for each subset of users assigned to time slot $t \in \mathcal{T} = \{1, ..., T\}$. Note that, in principle, each subproblem could be solved sequentially or in parallel using any algorithmic procedure. As in our case each subproblem is formulated as an ILP problem, so far now, we solve its linear programming (LP) relaxation to compute the bounds. In fact, this is a key aspect in our proposed VNS approach since the LP relaxations of the subproblems are very tight. Since each user must be attended by the BS in only one time slot $t \in \mathcal{T}$, the final solution of the problem can be easily reconstructed for the original problem from the solutions of each time slot $t \in \mathcal{T}$. The decomposition of the problem allows us to apply the VNS procedure in a straightforwardly manner and also to compute tight bounds easily. It turns out that solving the problem to optimality becomes rapidly prohibitive from a computationally point of view when the instances dimensions increase.

The paper is organized as follows. Section 2 briefly introduces the system description and presents the OFDMA-TDMA formulation of the problem. Section 3 presents the VNS algorithmic procedure while section 4 provides preliminary numerical results.

Finally, section 5 gives the main conclusions of the paper and provides some insights for future research.

2 Problem Formulation

We consider a BS surrounded by several mobile users within a single cell area. The BS has to assign a set of $\mathcal{N} = \{1, .., N\}$ subcarriers (or subchannels) to a set of $\mathcal{K} = \{1, .., K\}$ users in different time slots $\mathcal{T} = \{1, .., T\}$ in order to allow users to send signals to the BS. The allocation process is performed by the BS dynamically in time depending on the quality of the channels which are intrinsically stochastic. The latter affects the amount of bandwidth channel capacity needed by users to transmit their signals. Without loss of generality, we assume that the BS can fully and accurately predict the channel state information for each $t \in \mathcal{T}$. This is possible in OFDMA-TDMA networks when using adaptive overlapping pilots in uplink applications [11]. A scheduling formulation for an uplink wireless OFDMA-TDMA network can thus be written as follows

$$\mathcal{P}: \quad \max_{x,\varphi} \sum_{t=1}^{T}\sum_{k=1}^{K}\sum_{n=1}^{N} c_{k,n}^{t} x_{k,n}^{t} \tag{1}$$

$$\text{st:} \sum_{n=1}^{N} p_{k,n}^{t} x_{k,n}^{t} \leq P_k \varphi_{k,t}, \quad \forall k, t \tag{2}$$

$$\sum_{t=1}^{T} \varphi_{k,t} = 1, \quad \forall k \tag{3}$$

$$\sum_{k=1}^{K} x_{k,n}^{t} \leq 1, \quad \forall n, t \tag{4}$$

$$x_{k,n}^{t} \in \{0,1\}; \varphi_{k,t} \in \{0,1\}, \quad \forall k, n, t \tag{5}$$

where $x_{k,n}^{t}, \forall k, n, t$ and $\varphi_{k,t}, \forall k, t$ are the decision variables. These variables are defined as follows: $x_{k,n}^{t} = 1$ if user k is assigned subcarrier n at time slot t and zero otherwise. Similarly, $\varphi_{k,t} = 1$ if user k is scheduled to be attended in time slot t and zero otherwise. Matrices $(c_{k,n}^{t})$, $(p_{k,n}^{t})$ and (P_k) are input data matrices defined as follows. The entries in $(c_{k,n}^{t})$ denote the capacity achieved by user k using subcarrier n in time slot t while entries in $(p_{k,n}^{t})$ denote the power utilized by user k using subcarrier n in time slot t. Finally, (P_k) denotes the maximum power allowed for each user k to transmit their signals to the BS. The objective function in $\mathcal{P}$ is aimed at maximizing the total bandwidth channel capacity of the network. Constraint (2) is a maximum available power constraint imposed for each user k and for each time slot t to transmit signals to the BS. This is the main constraint which makes the difference between a downlink and an uplink process. In the former, there should be only one power constraint imposed for the BS whereas in the latter, each user is constrained by its own available maximum power $P_k, k \in \mathcal{K}$. Constraint (3) imposes the condition that each user must be attended by the BS in a unique time slot $t \in \mathcal{T}$. This constraint is specifically related to the time domain which is basically the transmission scheme of TDMA wireless networks. Whereas constraint (4) is related to the OFDMA scheme which imposes the condition

that each subcarrier should be assigned to at most one user at instant $t \in \mathcal{T}$. Finally, constraint (5) are domain constraints for the decision variables.

We note that $\mathcal{P}$ is an integer linear programming (ILP) formulation which is NP-Hard and thus difficult to solve directly for medium and large scale instances. Instead, we propose a VNS decomposition approach to compute tight bounds.

3 The VNS Approach

In order to compute tight bounds for $\mathcal{P}$ using a VNS metaheuristic approach, we first note that for any feasible assignment of $\varphi_{k,t} = (\bar{\varphi}_{k,t})$, i.e., such that $\sum_{t=1}^{T} \bar{\varphi}_{k,t} = 1, \forall k$. Problem $\mathcal{P}$ reduces to solving T subproblems of the following form

$$\mathcal{P}(t): \quad \max_{y} \quad \sum_{k \in \mathcal{K}_t} \sum_{n=1}^{N} \hat{c}^t_{k,n} y^t_{k,n} \tag{6}$$

$$\text{st:} \quad \sum_{n=1}^{N} \hat{p}^t_{k,n} y^t_{k,n} \leq \hat{P}_k, \quad \forall k \in \mathcal{K}_t \tag{7}$$

$$\sum_{k \in \mathcal{K}_t} y^t_{k,n} \leq 1, \quad \forall n \tag{8}$$

$$y^t_{k,n} \in \{0,1\}, \quad \forall k \in \mathcal{K}_t, n \in \mathcal{N} \tag{9}$$

where $\bigcup_{t=1}^{T} \mathcal{K}_t = \mathcal{K}$. Variables $y^t_{k,n}$ for each $k \in \mathcal{K}_t, n \in \mathcal{N}$ and $t \in \mathcal{T}$ are analogously defined as for $x^t_{k,n}$, i.e., $y^t_{k,n} = 1$ if user $k \in \mathcal{K}_t \subset \mathcal{K}$ is assigned subcarrier n in time slot t and zero otherwise. Matrices $(\hat{c}^t_{k,n})$, $(\hat{p}^t_{k,n})$, and $(\hat{P}_k)$ are respectively sub-matrices of $(c^t_{k,n})$, $(p^t_{k,n})$, and (P_k) we obtain from model $\mathcal{P}$ for each $t \in \mathcal{T}$ according to users in $\mathcal{K}_t$. Note that any solution $x^{t'}_{k,n}$ of $\mathcal{P}$ in a particular time slot $t' \in \mathcal{T}$ can be reconstructed by simply mapping the values of variables $y^{t'}_{k,n}$ $\forall k \in \mathcal{K}_{t'}, n \in \mathcal{N}$ into each user position in $x^{t'}_{k,n}$ $\forall k \in \mathcal{K}_{t'}$. All remaining values in $x^{t'}_{k,n}$ such that $k \notin \mathcal{K}_{t'}$ must be equal to zero. Therefore, for any feasible assignment $\varphi = \tilde{\varphi}$ the optimal solutions $\tilde{x}^t$ in $\mathcal{P}$ and optimal solutions $\tilde{y}^t$ in $\mathcal{P}(t)$, $\forall t \in \mathcal{T}$, we have

$$\sum_{t=1}^{T} \sum_{k=1}^{K} \sum_{n=1}^{N} c^t_{k,n} \tilde{x}^t_{k,n} = \sum_{t=1}^{T} \sum_{k \in \mathcal{K}_t} \sum_{n=1}^{N} \hat{c}^t_{k,n} \tilde{y}^t_{k,n} \tag{10}$$

Note that there are T^K feasible assignments for $\varphi_{k,t} = (\bar{\varphi}_{k,t})$ and each subset $\mathcal{K}_t$ has a cardinality of $\sum_{k \in \mathcal{K}} \bar{\varphi}_{k,t}$ users. In case any subset $\mathcal{K}_{t'} = \emptyset$, it means that no user is scheduled to be attended in time slot $t' \in \mathcal{T}$. Also notice that solving each $\mathcal{P}(t), \forall t \in \mathcal{T}$ such that $\mathcal{K}_t \neq \emptyset$ is an NP-Hard problem as it is equivalent to solve a multiple choice multiple knapsack problem [2].

VNS is a recently proposed metaheuristic approach [3] that uses the idea of neighborhood change during the descent toward local optima and to scape from the valleys that contain them. We define only one neighbor structure as $Ngh(\varphi)$ for $\mathcal{P}$ as the set of neighbor solutions φ' in $\mathcal{P}$ at a distance "h" from φ where the distance "h" corresponds to the number of users assigned in solutions φ' and φ. The VNS procedure we propose is depicted in algorithm 3.1. As input receives an instance of problem $\mathcal{P}$

Algorithm 3.1: VNS approach

Data: A problem instance of $\mathcal{P}$
Result: A tight solution $(\bar{x}, \bar{\varphi}, \bar{f})$ for $\mathcal{P}$
$Time \leftarrow 0$; $\mathcal{H} \leftarrow 1$; $count \leftarrow 0$; $\varphi_{k,n} \leftarrow 0, x^t_{k,n} \leftarrow 0, \forall k, n, t$;
foreach $k \in \mathcal{K}$ **do**
  choose randomly $t' \in \mathcal{T}$;
  $\varphi_{k,t'} \leftarrow 1$;
foreach $t \in \mathcal{T}$ **do**
  Solve the linear programming relaxation of $\mathcal{P}(t)$
Let $(\tilde{x}, \tilde{\varphi}, \tilde{f})$ be the initial solution found for $\mathcal{P}$ with objective value function $\tilde{f}$;
while *(*$Time \leq maxTime$*)* **do**
  for $i = 1$ *to* $\mathcal{H}$ **do**
    choose randomly $k' \in \mathcal{K}$ and $t' \in \mathcal{T}$;
    $\varphi_{k',t} \leftarrow 0, \quad \forall t \in \mathcal{T}$;
    $\varphi_{k',t'} \leftarrow 1$;
  foreach $t \in \mathcal{T}$ **do**
    Solve the linear programming relaxation of $\mathcal{P}(t)$
  Let (x^*, φ^*, g^*) be the new found solution for $\mathcal{P}$ with objective value function g^*;
  if *(*$g^* > \tilde{f}$*)* **then**
    $\mathcal{H} \leftarrow 1$;
    $(\tilde{x}, \tilde{\varphi}, \tilde{f}) \leftarrow (x^*, \varphi^*, g^*)$;
    $Time \leftarrow 0$; $count \leftarrow 0$;
  else
    Keep previous solution;
    $count \leftarrow count + 1$;
    if $\mathcal{H} \leq K$ *and* $count > \eta$ **then**
      $\mathcal{H} \leftarrow \mathcal{H} + 1$; $count \leftarrow 0$;
$(\bar{x}, \bar{\varphi}, \bar{f}) \leftarrow (\tilde{x}, \tilde{\varphi}, \tilde{f})$;

and provides a tight solution for it. We denote by $(\bar{x}, \bar{\varphi}, \bar{f})$ the final solution obtained with the algorithm where $\bar{f}$ represents the objective value function. The algorithm is simple and works as follows. First, it computes randomly a feasible assignment of $\tilde{\varphi} = (\tilde{\varphi}_{k,t})$ and solve each subproblem $\mathcal{P}(t), \forall t \in \mathcal{T}$ according to $\tilde{\varphi}$. This allows obtaining an initial solution $(\tilde{x}, \tilde{\varphi}, \tilde{f})$ for $\mathcal{P}$ that we keep. Next, the algorithm performs a variable neighborhood search by randomly scheduling $\mathcal{H} \leq K$ users in different time slots. Initially, $\mathcal{H} \leftarrow 1$ while it is increased in one unit when there is no improvement after new "η" solutions have been evaluated. On the other hand, if a new current solution found is better than the best found so far, then $\mathcal{H} \leftarrow 1$, the new solution is recorded and the process continuous. The whole process is repeated until the cpu time variable "$Time$" is less than or equal to the maximum available "$maxTime$". Note we reset "$Time = 0$" when a new better solution is found. This gives the possibility to search other "$maxTime$" units of time with the hope of finding better solutions.

As it can be observed, the VNS approach is constructed upon a key aspect of problem $\mathcal{P}$, namely its decomposition structure. On the other hand, the effectiveness of the algorithm also relies on the fact that the linear programming relaxation of each subproblem $\mathcal{P}(t), \forall t \in \mathcal{T}$ is very tight.

#	Instances Dimensions			Linear programs				VNS Approach			Gaps	
	K	N	T	$Opt.\mathcal{P}$	$\mathcal{LP}$	$Time\mathcal{P}$	$Time\mathcal{LP}$	$Ini.Sol.$	VNS	$Time$	$\mathcal{LP}$	VNS
1	8	32	10	2092	3488.1608	6.62	0.56	1622.7385	2118.9987	5.79	66.73	1.29
2	10	32	10	2771	3741.1867	13.28	0.62	2050.6937	2819.2678	7.49	35.01	1.74
3	12	32	10	3049	3967.9855	6.90	0.65	1786.4225	2972.4251	1.85	30.14	2.51
4	14	32	10	3109	4061.5510	8.79	0.71	2250.4192	3051.9827	1.71	30.63	1.83
5	20	32	10	3408	4137.6329	11.23	0.89	2246.0866	3409.5674	12.98	21.40	0.04
6	25	32	10	3591	4242.6719	28.84	1.00	2391.5610	3605.4858	11.20	18.14	0.40
7	30	32	10	3587	4208.7842	4.48	1.21	3039.4362	3592.3845	3.06	17.33	0.15
8	8	32	20	2351	6588.8126	11.48	0.84	1591.9860	2372.0500	1.45	180.25	0.89
9	10	32	20	2875	6548.8915	27.96	1.03	1505.5937	2879.8230	2.96	127.78	0.16
10	12	32	20	3281	7383.7370	51.51	1.11	2191.0147	3281.7037	1.60	125.04	0.02
11	14	32	20	4025	7801.8038	312.28	1.20	3077.3167	4025.6315	9.92	93.83	0.01
12	20	32	20	5965	8202.6180	74.56	1.51	3537.3049	5714.2134	58.03	37.51	4.20
13	25	32	20	6164	8195.6053	72.51	2.01	4022.6847	6186.9960	99.39	32.95	0.37
14	30	32	20	6548	8246.2234	105.09	2.28	4414.5224	6466.0896	115.84	25.93	1.25
15	8	64	10	3954	6754.0846	21.76	0.78	2829.6211	3970.8774	13.23	70.81	0.42
16	10	64	10	5606	7952.9545	35.12	0.87	3099.0138	5609.6752	6.56	41.86	0.06
17	12	64	10	5637	7780.5791	33.48	1.09	3950.4271	5541.4436	1.87	38.02	1.69
18	14	64	10	6334	7877.6160	47.54	1.20	5561.6601	6348.4896	3.87	24.37	0.22
19	20	64	10	6538	8225.0918	55.17	1.51	5386.4010	6553.5962	4.03	25.80	0.23
20	25	64	10	6941	8482.0337	77.23	1.78	6052.6174	6947.6831	5.95	22.20	0.09
21	30	64	10	7326	8496.0615	81.28	2.20	6282.7184	7326.0244	12.95	15.97	3e-4
22	8	64	20	4586	12789.0347	67.64	1.50	3200.9471	4544.7142	15.92	178.87	0.90
23	10	64	20	5753	14772.2571	167.57	1.60	4341.6136	5797.1178	10.15	156.77	0.76
24	12	64	20	6751	13449.2497	257.04	2.23	3891.1626	6781.0690	25.71	99.21	0.44
25	14	64	20	7692	14758.2530	576.20	2.37	4934.2025	7725.3751	18.18	91.86	0.43
26	20	64	20	11520	16342.8073	949.50	2.95	7692.8888	10795.0753	39.93	41.86	6.29
27	25	64	20	12297	16036.8844	536.17	3.86	8692.1874	12314.8432	110.20	30.41	0.14
28	30	64	20	12981	16873.0624	624.00	4.36	9327.2974	13017.9855	46.52	29.98	0.28
29	8	128	10	9292	15469.9953	167.86	1.31	6093.5991	9008.2856	32.44	66.49	3.05
30	10	128	10	10416	14341.4803	409.72	1.69	5150.2308	10590.8256	7.83	37.69	1.68
31	12	128	10	12248	16081.9795	728.13	1.77	8734.9364	12332.6018	13.91	31.30	0.69
32	14	128	10	12454	16002.4214	273.94	2.11	9538.7185	12510.7866	35.45	28.49	0.46
33	20	128	10	13441	16831.7606	387.81	2.69	9426.4508	13525.7884	9.31	25.23	0.63
34	25	128	10	14211	17059.0616	89.80	3.36	11492.2275	14236.2739	24.95	20.04	0.18
35	30	128	10	14546	17237.8062	519.84	4.39	12087.5565	14628.2521	8.53	18.51	0.57
36	8	128	20	9485	26344.7369	288.05	2.73	6802.1687	9547.7500	45.30	177.75	0.66
37	10	128	20	10993	25420.2375	479.06	3.84	8000.8647	11244.5429	6.77	131.24	2.29
38	12	128	20	13252	29327.4069	1577.70	4.05	8246.3013	13440.7492	23.98	121.31	1.42
39	14	128	20	14344	29151.5630	2239.27	5.73	8248.1641	14349.1162	33.30	103.23	0.04
40	20	128	20	23355	33356.0498	8416.50	5.64	16501.7620	22609.7154	34.83	42.82	3.19
41	25	128	20	24769	32729.1363	4964.91	7.48	16515.5073	24252.8271	96.31	32.14	2.08
42	30	128	20	25475	33352.2393	6170.50	11.44	17457.1910	25512.9978	162.50	30.92	0.15
Minimum values				2092	3488.2	4.48	0.56	1505.6	2119	1.45	15.97	3e-4
Maximum values				25475	33356	8416.5	11.44	17457	25513	162.50	180.25	6.29
Average values				8690.8	13431	737.57	2.43	6077.8	8656.2	28.18	61.37	1.04

Table 1 Upper and Lower bounds for $\mathcal{P}$

4 Numerical Results

In this section, we present preliminary numerical results for problem $\mathcal{P}$ using the proposed VNS algorithm. We generate realistic power data using a wireless channel from [7] while we set the capacities $c^t_{k,n} = \mathcal{M}^t_{k,n}, \forall k, n, t$ where $\mathcal{M}^t_{k,n}$ represents an integer number of bits randomly and uniformly generated between $\{1, .., 10\}$. These number of bits are required in higher order $\mathcal{M}$-PSK or $\mathcal{M}$-QAM modulation transmission schemes [1]. Specially for multimedia applications where the users bit rate demands are significantly higher. So far, we assume that the bit rate demands are uniformly distributed. In a larger version of this work, we will also consider other distribution types. Finally, we set $P_k = 0.4 \cdot \sum_{n \in \mathcal{N}} p^1_{k,n}, \forall k \in \mathcal{K}$ and $\eta = 500$. A Matlab program is implemented using CPLEX 12 to solve problem $\mathcal{P}$ while we use MOSEK solver [5] to solve its linear programming relaxation we denote hereafter by $\mathcal{LP}$ and each linear programming relaxation $\mathcal{P}(t), \forall t \in \mathcal{T}$ within each iteration of the VNS algorithm. The numerical experiments have been carried out on a Pentium IV, 1 GHz with 2 GoBytes of RAM under windows XP. In table 1, column 1 gives the instance number and columns 2-4 give the instances dimensions. In columns 5-8, we provide the optimal solutions of $\mathcal{P}$, $\mathcal{LP}$, and the cpu time in seconds CPLEX needs to solve $\mathcal{P}$ and $\mathcal{LP}$, respectively.

Similarly, in columns 9-11, we present the initial solutions found with the algorithm 3.1, its best solution found and the cpu time in seconds the algorithm needs to reach that solution. Notice that this cpu time considers all the time spent when solving all the subproblems involved in the algorithm sequentially and not in parallel as it could be improved. In all our tests we set the maximum time available to $maxTime = 50$ seconds. We also mention that whenever the variable $Time$ reached this amount, it means the algorithm did not find any better solution within 50 seconds, therefore we subtract this amount to the complete registered time. The latter provides the exact cpu time the VNS approach needs to find the best solution found so far. Finally, in columns 12 and 13 we provide gaps we compute as $\frac{\mathcal{LP}-Opt.\mathcal{P}}{Opt.\mathcal{P}} * 100$ and $\frac{|VNS-Opt.\mathcal{P}|}{Opt.\mathcal{P}} * 100$, respectively. Additionally, the last three rows in table 1 provide minimum, maximum and average values for columns 5-13, respectively. From table 1, we mainly observe that the bounds obtained with the VNS approach are very tight when compared to those obtained with $\mathcal{LP}$. For example, the gaps are less than 1 % in about 66.6 % and less 2 % in about 83.3 % of the instances when using the VNS approach. This is confirmed by the total average gap which is 1.04 %. Whereas the gaps obtained with the $\mathcal{LP}$ are not tight when compared to the optimal solutions in all cases. Another

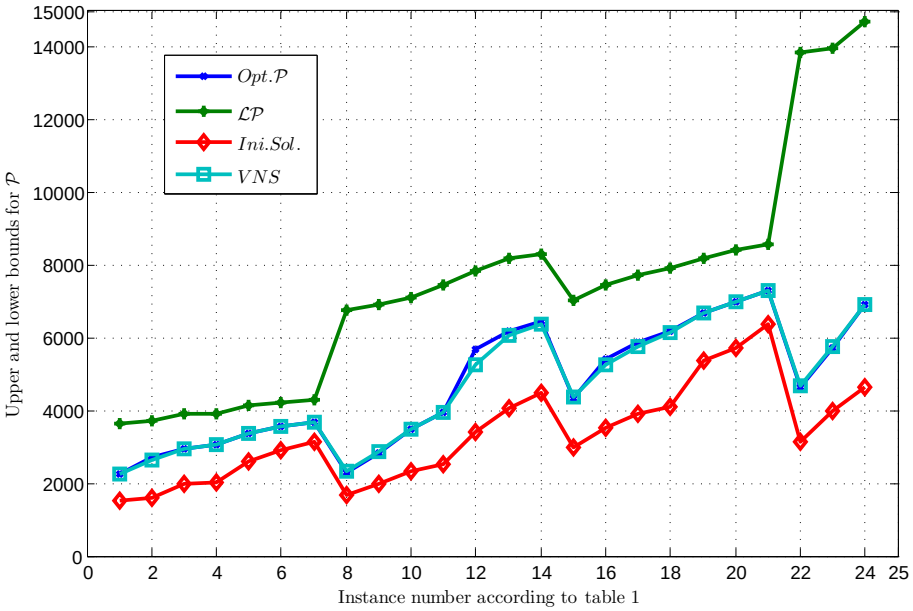


Fig. 1 Average bounds for instances 1-24 in table 1.

observation is that the average best solution found by the VNS algorithm improves in approximately 43% from the initial solution found by the algorithm which confirms its effectiveness. Moreover, when computing the difference for the average cpu time needed to solve problem $\mathcal{P}$ between the VNS approach and CPLEX, we obtain an improvement of 97,16 %. Finally, we observe that the cpu time required by CPLEX to compute an optimal solution of a particular instance grows rapidly while increasing its dimensions. So far, the averages presented in table 1 are computed using only one sam-

ple for the input data of instances 1-42. In order to provide more insight about these numerical results, in figures 1 and 2 we plot average results for instances 1-24 of table 1. We do not present averages for instances 25-42 since their cpu times become highly prohibitive as shown in table 1. For this purpose, we generate 25 samples for the input data of these instances. We use plots in this case to appreciate easily the trends of the average numerical results. In figure 1, the instance number appears in the horizontal axis while the vertical axis gives the averages we compute for the optimal solution of $\mathcal{P}$, for the linear programming relaxation of $\mathcal{P}$ ($\mathcal{LP}$), for the initial solution ($Ini.Sol.$) found with the VNS algorithm 3.1, and for the VNS approach respectively. Here, the trends of the curves mainly confirm the numerical results of table 1. We observe that VNS provides very tight near optimal solutions. By computing the average differences between VNS and the optimal solutions of $\mathcal{P}$ we obtain a 1.06 % of tightness which is similar to the average obtained in table 1. We also observe that the initial solutions are substantially improved by the VNS approach. In this case, we compute an average difference of 42.35% between the initials and best solutions of the VNS approach. Finally, we confirm that $\mathcal{LP}$ relaxation is not tight at all. In figure 2, the instance

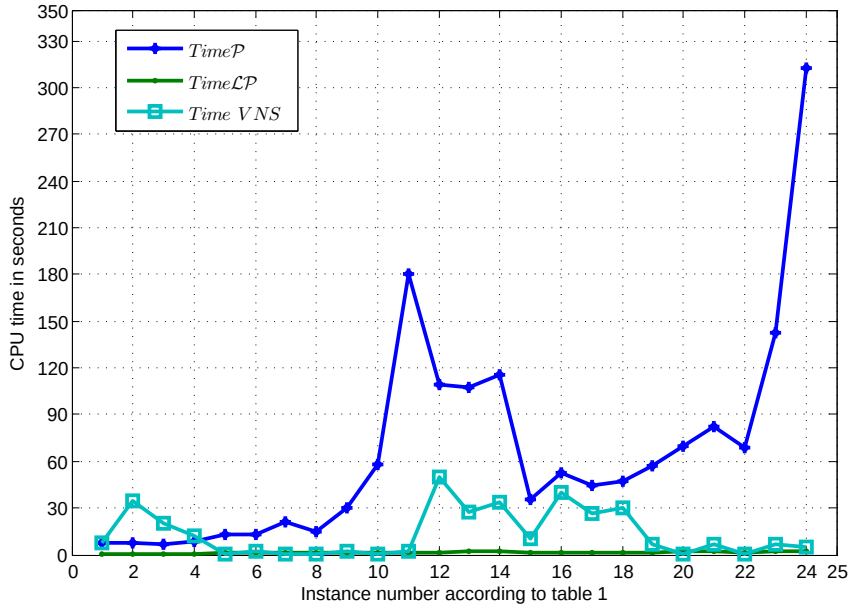


Fig. 2 Average CPU times in seconds for instances in table 1.

number appears in the horizontal axis while the vertical axis provides the average cpu time needed by CPLEX to solve problem $\mathcal{P}$, the average cpu time for $\mathcal{LP}$, and for the VNS approach as well. Here, we mainly observe that the cpu times required by VNS approach are significantly lower than CPLEX. In particular, we notice that for larger instances these cpu times remain below 10 seconds which is an interesting result.

5 Conclusions

In this paper, we proposed a hybrid resource allocation model for OFDMA-TDMA wireless networks and a VNS metaheuristic approach for solving the problem. The model is aimed at maximizing the total bandwidth channel capacity of an uplink OFDMA-TDMA network subject to user power and subcarrier assignment constraints while simultaneously scheduling users in time. As such, the model is best suited for non-real time applications where subchannel multiuser diversity can be further exploited in frequency and in time domains, simultaneously. The effectiveness of the proposed VNS approach relies on the decomposition structure of the problem which allowed solving a set of smaller integer linear programming subproblems within each iteration of the VNS approach. It turned out that the linear programming relaxations of these subproblems were very tight. Our numerical results showed tight bounds for the proposed algorithm, e.g. less than 2% in most of the instances. Besides, the bounds were obtained at a very low computational cost.

Future research will be focussed on developing other algorithmic approaches for solving each subproblem while considering other variants of the proposed model such as minimizing power subject to capacity constraints for uplink and downlink applications.

Acknowledgements The authors Pablo Adasme and Ismael Soto would like to thank to the projects: Conicyt/Insertion project Number 79100020, "Center for multidisciplinary research on signal processing" (project Conicyt/ACT1120) and project USACH/Dicyt Number 061117S) for their partial financial support.

References

1. Air Interface for Fixed Broadband Wireless Access Systems, MAC and Additional PHY Specifications for 2-11 GHz, IEEE Std. 802.16a, 2003.
2. Akbara Md Mostofa, M. Sohel Rahman, M. Kaykobad, E.G. Manning and G.C. Shoja, Solving the Multidimensional Multiple-choice Knapsack Problem by constructing convex hulls, Computers & Operations Research **33**, 12591273, (2006).
3. Hansen Pierre and Nenad Mladenovic, Variable neighborhood search: Principles and applications, European Journal of Operational Research **130**, 449-467 (2001).
4. Keivan Navaie and Halim Yanikomeroglu, Optimal Downlink Resource Allocation for Non-realtime Traffic in Cellular CDMA/TDMA Networks, IEEE Communications Letters, **10**, (2006).
5. MOSEK is an optimization software designed to solve large-scale mathematical optimization problems http://www.mosek.com/.
6. Sternad M., T. Svensson, T. Ottosson, A. Ahlen, A. Svensson and Anna Brunstrom, Towards Systems Beyond 3G Based on Adaptive OFDMA Transmission, Proceedings of the IEEE, 95, (2007).
7. Zukang Shen, Jeffrey G. Andrews and Brian L. Evans, Short Range Wireless Channel Prediction Using Local Information, Signals, Systems and Computers, In Conference Record of the Thirty-Seventh Asilomar Conference on, **1**, 1147-1151, (2003).
8. Yu-Jung Chang, Feng-Tsun Chien, and C. Kuo, Cross-layer QoS Analysis of Opportunistic OFDM-TDMA and OFDMA Networks, IEEE Journal on Selected Areas in Communications, Vol. 25, Issue 4, (2007), pp. 657-666.
9. M. Katoozian, Navaie, K., and Yanikomeroglu, H., Optimal Utility-Based Resource Allocation for OFDM Networks with Multiple Types of Traffic, In IEEE Vehicular Technology Conference, 2008. VTC Spring 2008, pp. 2223-2227.
10. K. Navaie and H. Yanikomeroglu, Optimal Downlink Resource Allocation for Non-realtime Traffic in Cellular CDMA/TDMA Networks, IEEE Communications Letters, vol. 10, pp. 278280, April 2006.

11. Mikael Sternad and Daniel Aronsson, Channel estimation and prediction for adaptive OFDMA/TDMA uplinks based on overlapping pilots, In Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2005, Vol. 3, pp. 861–864.

MISTA 2013

Bilevel Decomposition Approach for Cyclic Staff Scheduling Problems

Tatsushi Nishi · Taichi Sugiyama

Abstract In this paper, a bilevel decomposition approach for cyclic staff scheduling problem is presented. A cyclic staff scheduling problem is to create a set of cyclic schedule with several workforce regulations. The objective is to minimize the weighted sum of the total days and the deviation from average working time for fair working condition. The original problem is decomposed into the master problem and the sub-problems for each cyclic schedule. The master problem formulated as an unrelated parallel machine scheduling problem to minimize makespan is solved by an efficient branch and bound algorithm. The cuts are added to improve the lower bound. The proposed approach is applied to a railway crew rostering problem. The performance of the proposed method is compared with the general-purpose solvers with the constraint programming method. Computational results show the effectiveness of the proposed method with small duality gap.

1 Introduction

Cyclic staff scheduling problem is a rotating workforce scheduling problem that has high relevance on a broad range of work places in several industries. The problem asks to determine a set of rosters which is a cyclic sequence of daily shifts when a set of duties is given. Each duty has information on starting time and completion time. The aim is to find an assignment of duties and cyclic schedules that satisfies several workforce requirement.

Several methods have been studied for solving the problems in the literature. Especially, the problem has been studied on railway crew rostering problems[1], [2]. Heuristic approaches [3], [4], [5], [6], [7], [8], and meta-heuristic approaches [9], [10], [11], [12], [13] have been widely studied. However, the heuristic methods have a disadvantage that the optimality of solution cannot be ensured. Column generation algorithms are widely used to solve the problem [14], [15], [16], [17], [18]. The column generation method is effective to find an approximate solution for large scale problem. However, it is difficult

Tatsushi Nishi Taichi Sugiyama
Mathematical Science for Social Systems
Graduate School of Engineering Science, Osaka University
E-mail: nishi@sys.es.osaka-u.ac.jp

to consider min-max type of objective functions. Constraint programming techniques are also applied to solve the problems [17], [19], [20], [21],

Form the problem formulation of rostering problems, set-partitioning model [22] and 0-1 multi-commodity flow model [23] have been addressed. The fair workforce condition is one of the most important indicators of objective function. Several works have addressed the importance of the objective of fair workforce condition [2], [4]. A bilevel decomposition algorithm for railway crew rostering problem has been proposed [26]. The main difference with the previous work is that the cell-based formulation is utilized to formulate the original problem without graph-based formulation [26]. The original problem is decomposed into an upper level master problem and lower level subproblems. The upper level master problem determines an assignment of duties to the set of rosters. The lower level subproblem generates a feasible cyclic sequence with several workforce regulations. The effective cuts are embedded into the master problem to reduce the feasible search space to tighten the duality gap between the solutions of the bilevel program.

In this paper, we present an improved algorithm for bilevel decomposition approach to cyclic staff scheduling problem for fair workforce condition. The performance of the bilevel decomposition algorithm is improved from [26]. In order to improve the computational efficiency, the master problem is solved by an efficient branch and bound algorithm. The accuracy of lower bound is also improved by incorporating valid constraints further. Computational experiments are conducted to show the effectiveness of the proposed method comparing the solutions with those of the conventional algorithm and the constraint programming techniques.

1.1 Problem statement

A duty is a daily working plan for each staff. A roster is a cyclic sequence of duties. It represents a monthly working plan for a set of staffs. The cyclic staff scheduling problem is to find a set of rosters that is feasible cyclic sequence of duties when the number of rosters is given. The rosters are required to satisfy a number of constraints. Each duty identified by index i has start time S_i, finish time F_i and working duration W_i. Each duty i has an attribute whether or not a duty is overnight duty if it requires working during the overnight. We construct a set of several rosters to partition all of given duties. It is required to satisfy a number of constraints imposed by the regulation and labor laws. The constraints considered in this paper are listed as follows.

i) Each week has more than $D1$ rest days.
ii) The minimum resting time between the consecutive two duties is equal to the working time of a previous duty.
iii) The minimum resting time between two duties before and after a rest day is $H3$
iv) A finish time of a duty before a rest day is after $H4$.
v) A start time of a duty after a rest day is before $H5$.
vi) The consecutive overnight duties are at most two.
vii) The maximum days for each roster has is $D7$.

The performance indices consist of the fairness of labor conditions for rosters and the total cost of rosters. In order to equalize average working time, the maximum value of the average working time in a month for rosters is minimized. The objective function is the weighted sum of the maximum monthly average working time and the total number

of days. The empty day that has no duty is permitted. The empty day cannot be a rest day due to violations of constraints of iii)-v). An example of duties and rosters is shown in Fig. 1.

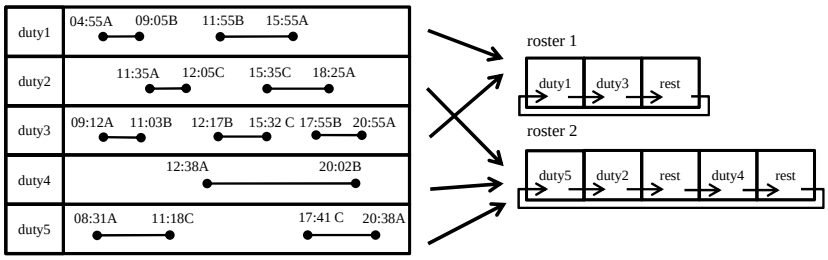


Fig. 1 Duty and roster

1.2 Modeling cyclic staff scheduling problem

Let I be the set of duties given and K be the set of rosters to be constructed. We assign each duty $i \in I$ to one of $|K| \times D7$ cells (Fig. 2). The rest days and last day which means the end of rosters (last days are rest days) are also determined.

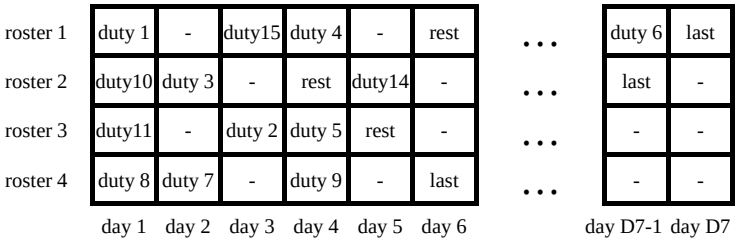


Fig. 2 Model for cyclic staff schedule

1.3 Problem formulation

Sets

I : Set of duties
K : Set of rosters
D : Set of days ($D = \{1, 2, ..., D7\}$)
I^M : Set of night duties ($I^M \subset I$)

Constants

S_i : the start time of duty $i \in I$

F_i : the finish time of duty $i \in I$
W_i : the working time of duty $i \in I$
$H0$: the constant value which indicates one hour
$H3$: the minimum rest between two duties
$H4$: the latest finish time of duty before a rest day
$H5$: the earliest start time of duty after a rest day
$D7$: the minimum rest days for a week

Decision variables

$$x_{i,k,d} = \begin{cases} 1 & \text{if duty } i \text{ is performed on day } d \text{ in roster } k \\ 0 & \text{otherwise} \end{cases}$$

$$h_{k,d} = \begin{cases} 1 & \text{if day } d \text{ in roster } k \text{ is the rest day} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{k,d} = \begin{cases} 1 & \text{if day } d \text{ in roster } k \text{ is the last day (rest day)} \\ 0 & \text{otherwise} \end{cases}$$

$\alpha \in \mathbb{R}$: the maximum value of monthly average working times

$$(IP) \quad \min \left(\mu_1 \alpha + \mu_2 \sum_{k \in K} \sum_{d \in D} d \; f_{d,k} \right) \tag{1}$$

s. t.

$$\sum_{i \in I} x_{i,k,d} + h_{k,d} + f_{k,d} \leq 1, \quad \forall d \in D, k \in K \tag{2}$$

$$\sum_{k \in K} \sum_{d \in D} x_{i,k,d} = 1, \quad \forall i \in I \tag{3}$$

$$\sum_{d=d'}^{d'+6} h_{k,d} + M \sum_{d=1}^{d'+6} f_{k,d} \geq D1, \quad \forall k \in K, d' = 1, 2, ..., D7 - 6 \tag{4}$$

$$\sum_{i \in I} S_i \; x_{i,k,d+1} + 24H0 - \sum_{i \in I} F_i \; x_{i,k,d} + M(1 - \sum_{i \in I} x_{i,k,d}) +$$
$$M(1 - \sum_{i \in I} x_{i,k,d+1}) \geq \sum_{i \in I} W_i x_{i,k,d}, \forall k \in K, d \in D \setminus \{D7\} \tag{5}$$

$$\sum_{i \in I} S_i \; x_{i,k,d+1} + 48H0 - \sum_{i \in I} F_i x_{i,k,d-1} + M(1 - h_{k,d})$$
$$+M(1 - \sum_{i \in I} x_{i,k,d+1}) + M(1 - \sum_{i \in I} x_{i,k,d-1}) \geq H3,$$
$$\forall k \in K, d \in D \setminus \{1, D7\} \tag{6}$$

$$\sum_{i \in I} S_i \; x_{i,k,1} + 48H0 - \sum_{i \in I} F_i x_{i,k,d-1} + M(1 - f_{k,d}) + M(1 - \sum_{i \in I} x_{i,k,1})$$
$$+M(1 - \sum_{i \in I} x_{i,k,d-1}) \geq H3, \; \forall k \in K, d \in D \setminus \{1, D7\} \tag{7}$$

$$\sum_{i \in I} F_i \; x_{i,k,d-1} - M(1 - h_{d,k}) \leq H4, \forall d \in D, \forall k \in K \tag{8}$$

$$\sum_{i \in I} F_i \; x_{i,k,d-1} - M(1 - f_{d,k}) \leq H4, \forall d \in D, \forall k \in K \tag{9}$$

$$\sum_{i \in I} S_i \; x_{i,k,d+1} + M(1 - h_{d,k}) + M(1 - \sum_{i \in I} x_{i,k,d+1}) \geq H5$$
$$\forall d \in D \setminus \{D7\}, \forall k \in K \tag{10}$$

$$\sum_{i \in I} S_i \; x_{i,k,1} \geq H5, \forall k \in K \tag{11}$$

$$\sum_{d=d'}^{d'+2} \sum_{i \in I^M} x_{i,k,d} \leq 2, \forall k \in K, d' = 1, 2, ..., D7 - 2 \tag{12}$$

$$\frac{28}{d} \sum_{i \in I} \sum_{d' \in D} W_i x_{i,k,d'} \leq \alpha + M(1 - f_{k,d}), \; \forall k \in K, \; d \in D \tag{13}$$

$$\sum_{d \in D} f_{k,d} = 1, \forall k \in K \tag{14}$$

$$f_{k,d} + \sum_{i \in I} x_{i,k,d'} + h_{k,d'} + f_{k,d'} \leq 0, \forall d \in D$$
$$d' = d, d+1, \ldots, D7, \forall k \in K \tag{15}$$

$$\sum_{d \in D} df_{k,d} - \sum_{d \in D} df_{k+1,d} \geq 0, \forall k \in K \setminus \{|K|\} \tag{16}$$

$$x_{i,k,d} \in \{0, 1\}, \; \forall i \in I, \; \forall k \in K, \; d \in D \tag{17}$$

$$h_{k,d}, \; f_{k,d} \in \{0, 1\}, \; \forall k \in K, \; d \in D, \alpha \in \mathbb{N} \tag{18}$$

Constraints (2) imply that a duty, a rest day, a last day or empty day is assigned to each cell of rosters. Constraints (3) ensure that each duty $i \in I$ is performed. Constraints (4) ensure that each week has more than $D1$ rest days. Constraints (5) indicate that the minimum resting time between the consecutive two duties is equal to the working time of a previous duty. Constraints (6) and (7) indicate that $H3$ of resting time is required between two duties before and after a rest day. Constraints (8) and (9) restrict that the finish time of duty before a rest day is before $H4$. Constraints (10) and (11) represent that the start time of duty after a rest day is after $H5$. Constraint (12) prohibits three consecutive night duties.

For each roster, constraints (13) are ignored if the value of d is not equal to the total days of the roster. If d is the total days of the roster, the left hand-side of the equation is equal to the average working time of the roster in terms of four weeks. Hence, α is the monthly maximum average working time when the objective function (1) is minimized. Constraints (14) mean that we one last day is assigned to each roster. Constraints (15) ensure that all days after the last day are empty days. Constraints (16) state that the rosters are sorted in descending order of the total days of each roster. These redundant constraints can decrease the size of the search space and reduce the computation time.

2 Bilevel decomposition approach

Because the objective function is to minimize the maximum value, it is extremely difficult for column generation method to solve the problem. In railway company, practitioners create the rosters at two stages. Firstly, a set of duties is divided into some

groups. Secondly, a set of rosters is generated to satisfy the constraints. From the idea, we propose a bi-level decomposition approach in order to reduce the computational effort. The upper level master problem determines an assignment of duties to the set of rosters and the lower level subproblem generates a feasible sequence of rosters including several workforce regulations.

In the proposed method, we decompose an original problem (IP) into an upper-level master problem and lower-level subproblems. The lower bound of the original problem is obtained from the solution of the master problem. The upper bound is obtained from the solutions of the subproblem where some decision variables of the master problem are fixed. These problems are iteratively solved. The cuts are embedded in the master problem in order to eliminate the non-promising candidates of solution.

2.1 Master problem

In the master problem, we relax the constraints iii)-v) related to the sequence of duties and the rest days. The master problem determines an assignment of crew duties to the set of rosters and the number of days of each roster in order to consider the maximum value of monthly average working time of rosters.

2.1.1 Formulation of master problem

Decision variables

$$\delta_{i,k} = \begin{cases} 1 \text{ if the duty } i \text{ is assigned to the roster } k \\ 0 \text{ otherwise} \end{cases}$$

$$f_{d,k} = \begin{cases} 1 \text{ if the total days of the roster } k \text{ is } d \\ 0 \text{ otherwise} \end{cases}$$

The formulation of the master problem ($IDP1$) is as follows.

$$(IDP1) \quad \min \quad \left(\mu_1 \alpha + \mu_2 \sum_{k \in K} \sum_{d \in D} d \; f_{d,k} \right) \tag{19}$$

s. t.

$$\sum_{k \in K} \delta_{i,k} = 1, \; \forall i \in I \tag{20}$$

$$\sum_{d \in D} f_{d,k} = 1, \; \forall k \in K \tag{21}$$

$$\sum_{d \in D} d \; f_{d,k} \geq \frac{7}{7 - D1} \sum_{i \in I} \delta_{i,k}, \; \forall k \in K \tag{22}$$

$$\frac{28}{d} \sum_{i \in I} W_i \delta_{i,k} \leq \alpha + M(1 - f_{d,k}), \; \forall k \in K, \; d \in D \tag{23}$$

$$\sum_{d \in D} d \; f_{d,k} \geq \sum_{d \in D} d \; f_{d,k+1}, \; \forall k \in K \setminus \{|K|\} \tag{24}$$

$$\delta_{i,k} \in \{0,1\}, f_{d,k} \in \{0,1\} \; \forall i \in I, \; \forall k \in K, \; d \in D \tag{25}$$

$$\alpha \in \mathbb{N} \tag{26}$$

The objective function (19) is the weighted sum of the maximum monthly average working time in terms of four weeks (α) and the total number of days ($\sum_{k\in K}\sum_{d\in D} d\ f_{d,k}$) and it means the equalization of the workforce conditions and the minimization of the total days. Constraints (20) ensure that each duty i is assigned to one roster. Constraints (21) state that each roster must finish within D days. From the constraints (6), $(7-D1)$ duties require $D1$ rest days. Therefore, the constraints (22) ensure that the number of days of each roster is larger than the minimum required days. The constraints (23) state that α is, similarly to (9), the maximum average working time in terms of four weeks. For each roster, the equation is ignored if $f_{d,k} \neq 1$. Constraints (24) state that the rosters are sorted in descending order of the total of days of each roster. We show the following proposition related to the master problem.

Proposition 1 *The master problem* $(IDP1)$ *is the relaxation of* (IP)*, and a lower bound of* (IP) *is obtained from the optimal value of* $(IDP1)$*.*

Proof Let $\delta_{i,k}$ be defined by

$$\delta_{i,k} = \sum_{d\in D} x_{i,k,d},\ \forall i \in I,\ \forall k \in K \tag{27}$$

$\delta_{i,k}$ is binary variable which indicates whether the duty i is assigned to the roster k. From (3) of (IP),

$$\sum_{k\in K} \delta_{i,k} = 1,\ \forall i \in I \tag{28}$$

The constraints (4) state that each week has more than $D1$ rest days. Therefore $(7-D1)$ duties require $D1$ rest days. Then the following constraints are always satisfied.

$$\sum_{d\in D} d\ f_{d,k} \geq \frac{7}{7-D1}\sum_{i\in I}\delta_{i,k},\ \forall k \in K \tag{29}$$

The master problem $(IDP1)$ is a relaxation of the original problem (IP) if (2), (5)-(12), (15) are eliminated from (IP) and (3), (4), (17) are replaced by (27)-(29).

2.1.2 Branch and bound

It is too difficult to solve the master problem due to the constraints (23). The master problem $(IDP1)$ is to minimize the maximum value of average working times. If the total days for each roster are fixed, and the constraints (22) are relaxed, the problem is identical to unrelated parallel-machine scheduling problem to minimize makespan. The problem is known to be NP-hard[24]. For these reason, the master problem is solved by a branch and bound method. In this algorithm, we firstly fix the variable $f_{d,k}$ which indicate the total days of the roster k. The outline of the branch and bound algorithm is explained as follows.

Branching
The nodes are branched by the number of days of each roster and the assignment of each roster to each duty. We firstly branch by the number of days of each roster.

Node priority
The priority of searching the nodes is defined. The first priority is to select a node that has the minimum lower bound. The second priority is to select a node that has the deepest node. The third priority is to select a node that has the least the number of days or the node that has the minimum makespan.
Constraints
The constraints are checked whether the constraint (22) and cuts (See section 3) are satisfied or not. If all constraints are satisfied, we generate the new branching nodes.

The algorithm requires huge memory to solve the master problem if the information of all nodes are kept during the search. In order to avoid the situation, we use a best-bound search based algorithm. In the algorithm, a pruning lower bound $LB*$ is defined. If the lower bound of a generated node is equal or more than $LB*$, the node is discarded from the search. The search is executed until a derived upper bound is equal to the lower bound. $LB*$ is firstly equal to 0 and it is increased if all nodes are pruned.

2.2 Subproblems

The subproblem is to create a feasible sequence of rosters with several workforce regulations on the condition that the assignment of duties and the total days is fixed. The problem is a feasibility problem without objective function and it can be decomposed into each roster. It can significantly reduce the computational effort. If a feasible solution cannot be derived, the constraints related to the total days are relaxed. Then a roster with less days can be generated. For that purpose, the objective function of the subproblem is to minimize the total days for each roster.

2.2.1 Formulation of the subproblem

The formulation of the subproblem for roster k is written as follows.

Sets
I_k : Set of duties assigned to the roster k in master problem ($\bigcup_{k\in K} I_k = I$)
I_k^M : Set of night duties assigned to the roster k in master problem ($I^M \subset I_k$)

Constant
$d*_k$: the total days of roster k which is determined by the master problem

$$(IDP2-k) \quad \min \quad \left(\mu_2 \sum_{k\in K}\sum_{d\in D} d\ f_{d,k}\right) \tag{30}$$

s. t.

$$\sum_{i\in I_k} x_{i,k,d} + h_{k,d} + f_{k,d} \leq 1, \quad \forall d \in D \tag{31}$$

$$\sum_{d\in D} x_{i,k,d} = 1, \quad \forall i \in I_k \tag{32}$$

$$\sum_{d=d'}^{d'+6} h_{k,d} + M \sum_{d=1}^{d'+6} f_{k,d} \geq D1, \quad d' = 1, 2, ..., D7-6 \tag{33}$$

$$\sum_{i\in I_k} S_i\ x_{i,k,d+1} + 24H0 - \sum_{i\in I_k} F_i\ x_{i,k,d} + M(1 - \sum_{i\in I_k} x_{i,k,d})$$
$$+M(1 - \sum_{i\in I_k} x_{i,k,d+1}) \geq \sum_{i\in I_k} W_i x_{i,d}, \forall d \in D \setminus \{D7\} \quad (34)$$

$$\sum_{i\in I_k} S_i\ x_{i,k,d+1} + 48H0 - \sum_{i\in I_k} F_i x_{i,k,d-1} + M(1 - \sum_{i\in I_k} x_{i,k,d-1})$$
$$+M(1 - h_{k,d}) + M(1 - \sum_{i\in I_k} x_{i,k,d+1}) \geq H3,\ d \in D \setminus \{1, D7\} \quad (35)$$

$$\sum_{i\in I_k} S_i\ x_{i,k,1} + 48H0 - \sum_{i\in I_k} F_i x_{i,k,d-1} + M(1 - \sum_{i\in I_k} x_{i,k,1})$$
$$+M(1 - f_{k,d}) + M(1 - \sum_{i\in I_k} x_{i,k,d-1}) \geq H3,\ d \in D \setminus \{1, D7\} \quad (36)$$

$$\sum_{i\in I_k} F_i\ x_{i,k,d-1} - M(1 - h_{k,d}) \leq H4, \forall d \in D \quad (37)$$

$$\sum_{i\in I_k} F_i\ x_{i,k,d-1} - M(1 - f_{k,d}) \leq H4, \forall d \in D \quad (38)$$

$$\sum_{i\in I_k} S_i\ x_{i,k,d+1} + M(1 - h_{k,d}) + M(1 - \sum_{i\in I_k} x_{i,k,d+1}) \geq H5$$
$$\forall d \in D \setminus \{D7\} \quad (39)$$

$$\sum_{i\in I_k} S_i\ x_{i,k,1} \geq H5 \quad (40)$$

$$\sum_{d=d'}^{d'+2} \sum_{i\in I_k^M} x_{i,k,d} \leq 2, d' = 1, 2, ..., D7 - 2 \quad (41)$$

$$\sum_{d=d*_k}^{D7} f_{k,d*_k} = 1 \quad (42)$$

$$f_{k,d} + \sum_{i\in I_k} x_{i,k,d'} + h_{k,d'} + f_{k,d'} \leq 0, \forall d \in D, d' = d, d+1, \ldots, D7 \quad (43)$$

$$x_{i,k,d} \in \{0,1\},\ h_{k,d},\ f_d \in \{0,1\},\ \forall i \in I_k,\quad d \in D \quad (44)$$

Constraints (31) mean that a duty, a rest day, a last day or empty day is assigned to roster k. Constraints (32) ensure that each duty $i \in I$ is performed. Constraints (33) ensure that each week has more than $D1$ rest days. Constraints (34) mean that the minimum resting time between the consecutive two duties is equal to the working time of a previous duty. Constraints (35) and (36) represent that $H3$ of resting time is required between two duties before and after a rest day. Constraints (37) and (38) indicate that the finish time of duty before a rest day is before $H4$. Constraints (39) and (40) indicate that the start time of duty after a rest day is after $H5$. Constraints (41) prohibit three consecutive night duties. Constraints (42) ensure that we assign one last day to roster k and the last day is after day $d*_k$. Constraints (43) ensure that all days after a last day are empty days.

The subproblem is decomposed into each roster. This problem can be solved by the branch and bound by CPLEX because the size of the subproblem is sufficiently small.

3 Cut generation

At each iteration, a new solution is searched at the master problem until the duality gap between the upper bound and the lower bound exists within a tolerance. The cuts are used to generate new solutions in a master problem. In this section, we propose effective cuts to reduce search space.

3.1 Integer cuts

The integer cut that excludes the previously obtained feasible configuration of the generated solutions from the master problem. We set the solution of the master problem at iteration r as $\Delta(r) = \{(i,k)|\delta_{i,k} = 1\}$ and $F(r) = \{(d,k)|f_{d,k} = 1\}$. The following constraints are embedded into the master problem at $(r+1)$ iterations.

$$\sum_{(i,k)\in\Delta(r)} \delta_{i,k} + \sum_{(d,k)\in F(r)} f_{d,k} \leq |I| + |K| - 1, \ \forall r \tag{45}$$

The left hand-side of (45) is more than $|I| + |K| - 1$ if and only if the solution and the previously obtained solution at iteration r are the same.

3.2 Infeasible assignment cut

It is possible that feasible solution cannot be derived under the assignment of duties to a roster given by the master problem if we relaxed the constraints related to the total days. In this case, the set of assignments to all rosters which contains the feasible assignment to a roster is also infeasible regardless of the assignment to other rosters. Therefore, it is effective to exclude such infeasible solutions. If we cannot solve subproblems at iteration r, we set $\Delta^{rst}(r) = \{\ i \mid \delta_{i,k} = 1\}$. We call it infeasible assignment cut.

$$\sum_{i\in\Delta^{rst}(r)} \delta_{i,k} \leq |\Delta^{rst}(r)| - 1, \ \forall k \in K, \ \forall r \tag{46}$$

3.3 Infeasible day cut

If we relax the condition that the number of days is d determined in the master problem and a feasible roster with the total day d^*, we can know that a feasible roster of the same assignment requires at least d^* days. Therefore, we use the following cuts. If we cannot solve the subproblems at iteration r, we set $\Delta^{*rst}(r) = \{\ i \mid \delta_{i,k} = 1\}$ and $d^{*rst}(r) = d^*$.

$$\sum_{i\in\Delta^{*rst}(r)} \delta_{i,k} + \sum_{d=1}^{d^{*rst}(r)-1} f_{d,k} \leq |\Delta^{*rst}(r)|, \ \forall k \in K, \ \forall r \tag{47}$$

Constraints (47) represent that if we generate a roster with the number of days $d^* > d$ from the assignment of duties, we do not let the days of a new assignment which contains the assignment of duties with $d' < d^*$.

3.4 Effectiveness of Cuts

The integer cut is not effective because the cut excludes a single solution from the master problem. On the other hand, the infeasible assignment cut and the infeasible day cut can exclude a lot of non-promising solutions in a master problem. These cuts can effectively reduce the feasible search space. It is clear that the infeasible assignment cut and the infeasible day cut also never exclude optimal solutions because the solutions excluded by these cuts are infeasible. Therefore, the optimality of the decomposition algorithm can be ensured because these cuts never exclude optimal solutions.

3.5 Local search

In order to improve the upper bound derived by solving the subproblems, a local search is utilized to improve an upper bound. If a feasible set of rosters are obtained by solving the subproblems at each iteration, a local search is started from the solution. In this search, a day is selected from rosters and insert it to the day to another day. It is repeated until the upper bound is not updated.

3.6 Overall algorithm

The overall algorithm of the proposed method is as follows.

Step 1 Initialization.
Step 2 Solve the master problem.
  Step 2-1 Generate an initial node P_0 and $A := \{P_0\}$.
  Step 2-2 If $A = \emptyset$, then $LB^* := LB^* + 1$ and go to Step 2-1. Otherwise go to Step 2-3.
  Step 2-3 Choose $P_j \in A$ according to node priority.
  Step 2-4 If P_j is a solution of the master problem (The depth is $|K| + |I|$), go to Step 2-7.
  Step 2-5 Compute a lower bound for P_j. If the lower bound is less than $LB*$, $A := A - \{P_j\}$.
  Step 2-6 The new branched nodes $P_{j1}, P_{j2}, \ldots, P_{jq}$ which satisfy all constraints are generated from a node P_j. $A := A \cup \{P_{j1}, P_{j2}, \ldots, P_{jq}\} - \{P_j\}$ and go to Step 2-2.
  Step 2-7 If the derived upper bound for master problem is LB^*, the algorithm of solving master problem is finished and go to Step 3. Otherwise $A := A - \{P_j\}$ and go to Step 2-2.
Step 3 Solve the sub problems with fixed assignment of duties and the total days of rosters.
Step 4 Local search is executed from the solution of a feasible solution.
Step 5 Convergence evaluation. If the lower bound is equal to the upper bound or both of them are not updated pre-determined number of times, the algorithm is finished.
Step 6 Cut generation. Then return to Step 2.

4 Computational results

4.1 Effectiveness of proposed method

To illustrate the effectiveness of the proposed method, several computational experiments are executed. An intel® Core^{TM} i7 CPU 2.93GHz processor with 2GB memory is used and the algorithm was coded in C++ language using Visual C++ 2010 Express for computations. A master problem and subproblems formulated as an integer programming problem and they are solved by using the general-purpose solver CPLEX 12.1 (IBM ILOG). We generate the set of duties as an input data and execute experiments for several cases with different numbers of duties and rosters. The computational results are summarized in Table 1. In this table, a case is represented by (number of duties and number of rosters). LB is the lower bound obtained in master problem and UB is the upper bound obtained in subproblems. α is the maximum value of monthly average working times and d is the total days of rosters. itr. is the number of iterations in decomposition method. time is the total computation time. The objective function is the weighted sum of the maximum monthly average working time (expressed in minutes) and the total days. The weight coefficients selected as $\mu_1 = 1$, $\mu_2 = 280$. The cost of one day corresponds to the cost of difference of 10 minutes between rosters. From the results of Table 1, it is observed that optimal solution is derived for case 1 because the lower bound is equal to the upper bound.

Table 1 Computational results of proposed method

no.	duties	rosters	LB	UB	itr.	time	α	d
1	15	2	13104	13104	140	28	6384	24
2	30	2	18960	19724	109	81	7123	45
3	30	3	18960	19852	120	114	6972	46
4	50	3	25765	27747	118	193	7587	72
5	60	4	28835	31791	66	236	7431	87
6	70	4	32035	35856	73	312	7296	102
7	80	6	35740	38456	59	3600	8216	108
8	100	6	41799	46094	12	3600	7734	137

We compare the computational results between each method with or without cuts in order to confirm the effectiveness of the proposed cuts. “NO-CUT” means that no cut is embedded and we solve the master problem and the subproblems once. “INT-CUT” means that only integer cut is embedded. “ALL-CUT” means that all the proposed cuts are embedded. Table 2 shows the results of these experiments. cuts is the embedded pattern of cuts.

The upper bound of (NO-CUT) is better than that of (INT-CUT) in all cases when the cut patterns (NO-CUT) and (INT-CUT) are compared. T he upper bound of (ALL-CUTS) is better than that of (INT-CUT) in case 4 - 8. The results indicate that the integer cut is effective and embedding the infeasible assignment cut and the infeasible day cut can improve the performance of the decomposition method.

Table 2 Effects of cuts to performance of the proposed method

case	ALL-CUTS			NO-CUT			INT-CUT		
no.	LB	UB	t-time	LB	UB	t-time	LB	UB	t-time
1	13104	13104	28	13101	13114	0	13104	13104	28
2	18960	19724	79	18960	19725	0	18960	19724	81
3	18960	19852	114	18960	19989	1	18960	19852	112
4	25765	27747	193	25765	28424	1	25765	**27736**	208
5	28835	**31791**	236	28835	31796	2	28835	31792	51
6	32035	**35856**	312	32035	35858	2	32035	35857	141
7	35740	**38456**	3600	35740	38909	11	35740	38461	623
8	41799	**46094**	3600	41799	46804	967	41799	46756	1930

4.2 Comparison with the conventional method

In order to compare the proposed method with the conventional method, the comparison with solving the original problem by an integer programming solver (CPLEX) and a constraint programming (CP) solver are also executed. Since the lower bound is not obtained for CP, we compare only the upper bound of the proposed method with that of the conventional method. ($CPLEX$) is used to formulate IP. The formulation referring to the literature is used as the constraint programming formulation. CPLEX 12.1 (IBM ILOG) is used to solve IP and CP Optimizer (IBM ILOG) for constraint programming. The environments are the same with the previous experiment. Tables 3 shows the comparison of the proposed method with CPLEX (IP) and constraint programming (CP). The maximum time limit of 1 hour was fixed for CPLEX solver. We show the upper bound (UB) and the lower bound (LB) in some cases. The CPLEX solver fails to provide the optimal solution for case 6 and 8 within 1 hour. The results of proposed method are not better than that of CP for case 1- 5. However, the solution by proposed method is better than the solution by CPLEX in case 6 - 8. The proposed method is effective for the large scale problems because the size of case 6 - 8 is larger than the other cases.

Since the termination condition cannot be set for CP, a maximum time limit of 1 hour was fixed and we show the upper bound (UB) for CP solver. ( ) in time means the time when the upper bound is obtained for CP. The result of proposed method is not better than that of CP for all cases.

Table 3 Comparison of the proposed method (PM) and Integer Programming (IP)

case			Proposed			CPLEX			CP	
no.	duty	roster	LB	UB	t-time	LB	UB	t-time	UB	t-time
1	15	2	13104	13104	28	13104	13104	2053	13104	(87)
2	30	2	18960	19724	79	**19495**	19724	(3285)	19849	(591)
3	30	3	**18960**	19852	114	11630	**19724**	(3471)	19852	(1470)
4	50	3	**25765**	27747	193	19609	**27392**	(2359)	29815	(3597)
5	60	4	**28835**	31791	236	23554	**31598**	(3400)	33207	(1598)
6	70	4	**32035**	**35856**	312	23011	-	(3562)	36928	(2514)
7	80	6	**35740**	**38456**	3600	26451	41961	(3536)	41727	(1621)
8	100	6	**41799**	**46094**	3600	32834	-	(3496)	50498	(1649)

5 Conclusions

In this paper, we have proposed a bi-level decomposition approach to solve the cyclic staff scheduling problem with the objective of fair workforce condition. In the proposed approach, the original problem is decomposed into upper-level master problem and a lower-level subproblem. The upper-level master problem determines an assignment of duties to the set of rosters and the lower-level subproblem generates a feasible sequence of rosters including several workforce regulations. The effective cuts are proposed to reduce the feasible search space to tighten the duality gap. Computational results demonstrate the advantage of the proposed method and the effectiveness of the proposed cuts. In the future works, more effective cuts should be developed. We will apply this proposed approach to the other types of staff scheduling problems.

References

1. Sodhi, M.S., Norris, S., 2004. A flexible, fast, and optimal modeling approach applied to crew rostering at London underground. Annals of Operations Research 127, 259-281.
2. Hartog, A., Huisman, D., Erwin, J.W.A., Kroon, L., 1998. Decision support for crew rostering at NS. Public Transport 1, 121-133.
3. Bianco, L., Bielli, M., Mingozzi, A., Ricciardelli, S., Spadoni, M., 1992. A heuristic procedure for the crew rostering problem. European Journal of Operational Research 58, 272-283.
4. Maenhout, B., Vanhoucke, M., 2010. A hybrid scatter search heuristic for personalized crew rostering in the airline industry. European Journal of Operational Research 206, 155-167.
5. Sahin, G., Yuceogu, B., 2011. Tactical crew planning in railways. Transportation Research Part E 47, 1221-1243.
6. Caprara, A., Toth, P., Vigo, D., Fischetti, M., 1998. Modeling and solving the crew rostering problem. Operations Research 46(6), 820-830.
7. Zeghal, F.M., Minoux, M., 2006. Modeling and solving a crew assignment problem in air transportation. European Journal of Operational Research 175, 187-209.
8. Teodorovic, D., Lucic, P., 1996. A fuzzy set theory approach to the aircrew rostering problem. Fuzzy Sets and Systems 95, 261-271.
9. Moudani, W.El., Cosenza, C.A.N., de Coligny, M., 1993. A bi-criterion approach for the airlines crew rostering problem. Lecture Notes in Computer Science 1993, 486-500.
10. Santona, B., Sunarto, A., Rahman, A., 2010. Using differential evolution method to solve crew rostering problem. Applied Mathematics, 1, 316-325.
11. Lucic, P., Teodorovic, D., 1999. Simulated annealing for the multi-objective aircrew rostering problem. Transportation Research Part A 33, 19-45.
12. Souai, N., Teghem, J., 2009. Genetic algorithm based approach for the integrated airline crew-pairing and rostering problem. European Journal of Operational Research 199, 674-683.
13. Xie, L., Kliewer, N., Suhl, L., 2012. Integrated driver rostering problem in public bus transit. Procedia - Social and Behavioral Sciences.
14. Nishi, T., Muroi, Y., Inuiguchi, M., 2011. Column generation with dual inequalities for railway crew scheduling problems. Pubric Transport 3, 25-42.
15. Gamache, M., Soumis, F., Marquis, G., Desrosiers, J., 1999. A column generation approach for large-scale aircrew rostering problems. Operations Research 47(2), 247-263.
16. Freling, R., Lentink, R.M., Wagelmans A.P.M., 2004. A decision support system for crew planning in passenger transportation using a flexible branch-and-price algorithm. Annals of Operations Research 127, 203-222.
17. Sellmann, M., Zervoudakis, K., Stamatopoulos, P., Fahle, T., 2002. Crew assignment via constraint programming: integrating column generation and heuristic tree search. Annals of Operations Research 115, 207-225.
18. Saddoune, M., Desaulniers, G., Elhallaoui, I., Soumis, F., 2011. Integrated airline crew scheduling: a bi-dynamic constraint aggregation method using neighborhoods. European Journal of Operational Research 212, 445-454.

19. Medard, C.P., Sawhney, N., 2007. Airline crew scheduling from planning to operations. European Journal of Operational Research 183, 1013-1027.
20. Dawid, H., Konig, J., Strauss, C., 2001. An enhanced rostering model for airline crews. Computers & Operations Research 28, 671-688.
21. Caprara, A., Facacci, F., Lamma, E., Mello, P., Milano, M., Toth, P., Vigo, D., 1998. Integrating constraint logic programming and operations research techniques for the crew rostering problem. Software-Practice and Experience 28(1), 49-76.
22. Ryan, D.M., 2002. The solution of massive generalized set partitioning problems in aircrew rostering. Operations Research 43(5), 459-467.
23. Cappanera, P., Gallo, G., 2004. A multicommodity flow approach to the crew rostering problem. Operation Research 52(4), 583-596.
24. Lin, C., Liao, C., 2008. Makespan minimization for multiple uniform machines. Computers & Industrial Engineering 54, 983-992.
25. Mireault, P., Orlin, J.B., Vohra, R.V., 1997. A Parametric worst case analysis of the LPT heuristic for two uniform machines. Operational Research 45(1), 116-125.
26. Sugiyama, T., Nishi, T., Inuiguchi, M., Takahashi, S., Ueda, K., 2012. A bilevel decomposition approach to railway crew rostering problems for fair labor condition. Proc. IEEE International Conference on Automation Science and Engineering, 383-389.

MISTA 2013

Mitigating Partitioning Skew in MapReduce Computations

J. Berlińska · M. Drozdowski

Abstract In this paper we analyze handling partitioning skew in MapReduce computations. The basic MapReduce implementations strongly depend on the assumption that the data is partitioned evenly for reducing. However, in practical applications the data distribution is often skewed, what leads to decreasing MapReduce system performance. Using divisible load theory we analyze two methods of handling data skew in MapReduce computations. The proposed algorithms are evaluated in a series of computational experiments. To our best knowledge this is the first analytical study comparing mitigation of partitioning skew in two different stages of MapReduce applications.

1 Introduction

MapReduce is a parallel programming model for processing large data sets on big numbers of computers [10,17]. A MapReduce application consists of two stages: mapping and reducing. In the first stage a user-defined *Map* function processes the input dataset (e.g. a text or HTML file), and generates a set of intermediate $(key1, value1)$ pairs. In the second step the intermediate pairs are sorted by $key1$, and a *Reduce* function merges pairs with equal values of $key1$, to produce a list of result pairs $(key1, value2)$. As an example, consider constructing the inverted index for a set of documents, where all documents comprising certain *words* must be found [10]. The *Map* function parses the documents and generates a sequence of pairs $(word, docID)$, where $docID$ is a document identifier (e.g., a URL of a web page). The *Reduce* function accepts all $(word, docID)$ pairs containing a given $word$, sorts them and returns a $(word, list_docIDs)$ pair, where $list_docIDs$ is a sorted sequence of $docIDs$.

Both map and reduce operations are performed in parallel in a distributed computer system. The intermediate data obtained by a computer in the mapping stage are

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

divided into many parts, which are then assigned to the reducing processors by hashing. The obtained partitioning of the intermediate data has a big impact on the whole MapReduce application performance [13–15]. In the ideal case all reducing processors should finish computations at the same time. In reality the data distribution is often skewed and some processors work longer than others. Consequently, the total running time of the application increases. In this work we analyze this problem and present algorithms to solve it. Although practical methods of handling skew in MapReduce have been proposed earlier [13–15], to our best knowledge this is the first theoretical study dedicated to comparing approaches mitigating the skew in different stages of computations.

We propose a mathematical model of MapReduce based on the divisible load theory. Divisible load theory (DLT) was introduced in papers [1,9]. It is a model of distributed processing which assumes that the data to be processed, called load, can be continuously divided into pieces. There are no precedence constraints between these pieces, so that they can be processed independently on remote computers. Divisible load theory covers scheduling problems and performance modeling for various types of interconnection networks [9,11,8], systems with memory limits [16,3,5], and other. Surveys of divisible load theory, including applications, can be found e.g. in [2,7,12,18]. A divisible load model of MapReduce applications has been first proposed in [4,6]. In this work this model is expanded by relaxing the assumptions about the organization of communication between computers. We also provide a more detailed view of the reducing stage.

The rest of this paper is organized as follows. In Section 2 we present a mathematical model of MapReduce computations. The methods of handling partitioning skew in MapReduce applications are described in Section 3. The algorithms are compared experimentally in the following section. The last section is dedicated to conclusions.

2 Mathematical Model of MapReduce

In this section we present the mathematical model of MapReduce computations.

We assume that the application is run by a homogeneous computer system. We will use the terms computer, processor and worker interchangeably. Each computer comprises a CPU, some memory and an independent network interface (e.g. NIC and DMA). A processor can open at most one communication channel at a time, i.e. the so-called one-port model is used. The communication speed for two computers communicating in otherwise unused network is $1/C$. However, the bandwidth limit for concurrent channels in the whole network is l/C, so that at most l communication channels can be opened simultaneously without reducing the communication speed. We will call l the bisection width limit. If $n > l$ communications take place at the same time, then the bandwidth is shared equally between them, and the communication speed decreases to $l/(nC)$.

The total amount of load (input data) to be processed is V (e.g. bytes).

The execution of a MapReduce application begins with starting many copies of the program on a cluster of computers. A master computer assigns work to the other computers (workers). There are m processors executing map tasks (the mappers) and r processors executing reduce tasks (the reducers). The master starts processing a MapReduce application with dividing the input data into so-called splits and assigning them to the mappers. As the number of load splits is very big [10], the granularity of

the input load is fine. Hence, according to the methodology of DLT, we assume that the load is arbitrarily divisible. Each mapper obtains the same amount of data.

A mapper reads the assigned input load splits and processes the data using the Map function. We perceive these operations as processing with rate A (e.g. sec/bytes) [4,6]. Each mapper receives load of total size V/m and processes it in time AV/m. The results of mapping are divided into r parts by the partitioning function (usually of the form $hash(key1) \bmod r$) and written to r files on the local disk. Each file is dedicated to a particular reducer. We assume that the amount of results produced by mappers is proportional to the input size. For α bytes of input, $\gamma\alpha$ bytes of output are produced. The information about the locations of the files containing the intermediate results is sent back to the master. We will denote the size of a single piece of information, such as the location of a file, by ε. Thus, the communication of all mappers with the master takes time $Cmr\varepsilon$, as each of the m mappers created r files. The master forwards it to the reducers (again in time $Cmr\varepsilon$, because each of the r reducers obtains the locations of m files).

When a reducer receives the information about the files locations, it reads the buffered data from the local disks of the mappers. In order to guarantee that no computer performs two communications at the same time, a special communication organization is necessary. If $m \leq r$, then mapper i communicates consecutively with reducers $i, i+1, \ldots, r, 1, \ldots, i-1$. If $m > r$, then reducer j communicates consecutively with mappers $j, j+1, \ldots, m, 1, \ldots, j-1$. Each communication starts without unnecessary delay and no computer performs more than one communication at the same time. In the ideal case, when parts of data sent from mapper i to reducer j are equal for all i, j, there are always $\min\{m, r\}$ communications taking place at a given time, and the total time needed for sending load from the mappers to the reducers is $\max\{C, C\min\{m,r\}/l\}\gamma V/\min\{m,r\}$.

After reading all intermediate data, a reducer sorts it by the intermediate keys in order to group together all occurrences of the same intermediate key. The time needed to sort the data of size x is $a^{sort} x \log x$. Each key and the corresponding set of values are then processed by the Reduce function. The computing rate of a reducer will be denoted by a^{red}. Thus, the reducer execution time for input data of size x is $a^{red}x$. The output generated for each intermediate key is appended to a final output file for a given reducer. Thus, the final output of MapReduce is available in r output files located on the reducers.

In the ideal case, the partitiong function divides the space of intermediate keys so that each reducer obtains from each mapper data of the same size $\gamma V/(mr)$, i.e. the total load received by any reducer is $\gamma V/r$. Then, the running time of the MapReduce application is

$$T^* = AV/m + 2Cmr\varepsilon + \max\{C, C\min\{m,r\}/l\}\gamma V/\min\{m,r\} + a^{sort}(\gamma V/r)\log(\gamma V/r) + a^{red}\gamma V/r. \quad (1)$$

However, in reality the load partition is not ideally balanced and some reducers receive more data than the others. Such *partitionig skew* may lead to a performance drop, as the execution of the whole application finishes when the last reducer completes computations. The running time of the application depends in this case on many factors and cannot be presented by a simple formula similar to (1).

3 Algorithms

In this section we present two algorithms for mitigating partitioning skew in MapReduce: a static and a dynamic method. The static approach to the problem is to improve the load partitioning before sending the data from the mappers to the reducers. In the dynamic methods, the load distribution between the reducers can only be changed after the reducers obtained the data and started processing.

3.1 Static Method

A static method of load balancing, called fine partitioning, was proposed in [13]. The idea was to change the partitioning function so that it divides the space of the key values not into r parts, but into kr parts for some $k > 1$. This allows to decrease the impact of creating parts of different sizes by appropriately assiging the parts to the reducers. After the mappers finish their computations, they send the information about the obtained part sizes to the master computer. In order to obtain the optimum load assignment for the reducers, it is necessary to solve an NP-hard bin-packing problem. Therefore, the master uses heuristic approach similar to LPT algorithm. It sorts the partitions according to non-increasing sizes and assigns them one by one to the reducers with the smallest total load. The reducers obtain the information about the partitions they are to process, read the data from the mappers and process them.

We will now present a divisible load model of MapReduce with fine partitioning. In the first step, all mappers read and process the load with rate A, what takes time AV/m. Afterwards, each of the mappers sends the information about the sizes of kr partitions it created to the master. As we use 1-port model, the communications with the master must be sequential and the time necessary to complete them is $Ckrm\varepsilon$. The master computes the total sizes of the partitions, sorts them and assigns to the reducers, what can be done in total time $t^{master} = a^{master}(krm+kr\log(kr)+kr\log r)$, where a^{master} is the computing rate of the master. The information about the assignment is then sent sequentially to the reducers, in time $Ckrm\varepsilon$. Next, the reducers start reading the data from the mappers. After obtaining its load of size $\delta_j\gamma V$ (where $\sum_{j=1}^{r}\delta_j = 1$), reducer j sorts the data and performs reducing, in total time $a^{sort}(\delta_j\gamma V)\log(\delta_j\gamma V)+a^{red}\delta_j\gamma V$.

3.2 Dynamic Method

A system allowing for MapReduce skew mitigation in a dynamic way, called SkewTune, was described in [15]. The goal of SkewTune was to handle different types of skew, in both mapper and reducer stage of computations. As partitioning skew only is the subject of this work, we propose here a simpler approach. While SkewTune tries to balance the load distribution each time when a reducer finishes computations, we stop the computations to redistribute the load only once.

In our algorithm mapping and sending load from the mappers to the reducers proceeds as described in Section 2. Each reducer starts sorting (and then reducing) as soon as it receives all data. It notifies the master when it finishes sorting and when it finishes reducing. As soon as all reducers completed sorting and at least one reducer processed all its load, the master sequentially stops the r_1 reducers which still have some load to process, in time $Cr_1\varepsilon$. Afterwards, each of these reducers informs the

master about the amount of its remaining load, in total time $Cr_1\varepsilon$. Based on this data, the master computes the expected time of finishing the application. Then, it analyzes the scenario in which the most loaded reducer sends part of its remaining data to the least loaded reducer (which already finished processing its part), so that they finish computations at the same moment. According to the methodology of DLT, the master assumes that it is possible to divide the load into parts of any sizes, although in practice some rounding may be necessary, as records with the same key must be processed by the same reducer. If balancing the load assignment in this way is profitable, the master checks the next scenario, in which two most loaded processors hand over parts of their data to the least loaded reducers. This process continues as long as it is possible to decrease the total processing time. The running time of master computations is proportional to the number of analyzed cases. After choosing the best option, the master notifies the reducers to resume computing and informs them about the load parts they should send to different computers. The reducers perform the operations indicated by the master, rounding the load sizes if necessary. The execution of the application is completed when the last reducer finishes computing.

4 Computational Experiments

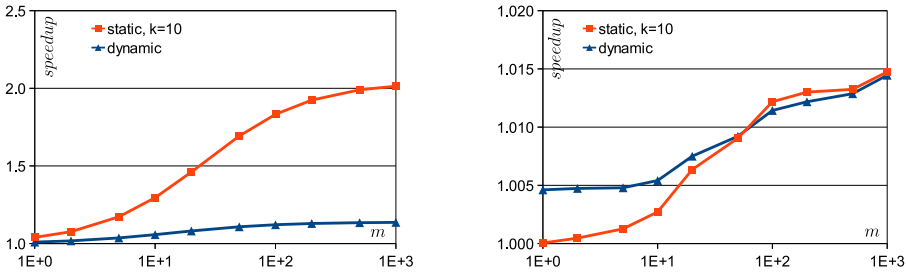
In this section we analyze the influence of system parameters on the improvement of MapReduce performance achieved by the proposed algorithms. In all test instances the number of keys in the input data is 1E6. The key frequencies are obtained in the following way. For each key i a number f_i is selected randomly with uniform distribution from interval $(1-\Delta, 1+\Delta)$. The frequency of i is set to $f_i / \sum_j f_j$. Thus, the key frequencies are not very unbalanced, as the maximum possible value is $1+\Delta$ times greater than the average.

Two types of instances will be analyzed. In the easy instances the partitioning function is consctructed by assigning the keys to the reducers randomly in such a way that each reducer receives the same number of keys. Hence, the load partitioning for the reducers will be rather balanced. In the hard instances each reducer obtains the same number of keys, but reducer 1 receives the keys with the greatest frequencies, reducer 2 the remaining keys with the greatest frequencies, and so on. Thus, the load distribution will be unbalanced.

Unless stated otherwise, we assume the following parameter values: V = 1E12, $\varepsilon = 8$, $m = 1000$, $r = l = 100$, $\gamma = 0.1$, C = 1E-8, $A = a^{master} = a^{sort}$ = 1E-7, a^{red} = 1E-6, $\Delta = 1$. The reducing rate a^{red} is greater than the remaining computation rate parameters in order to emphasize the reducing phase, for which load balancing is crucial. The number of partitions per reducer created by the static algorithm is $k = 10$.

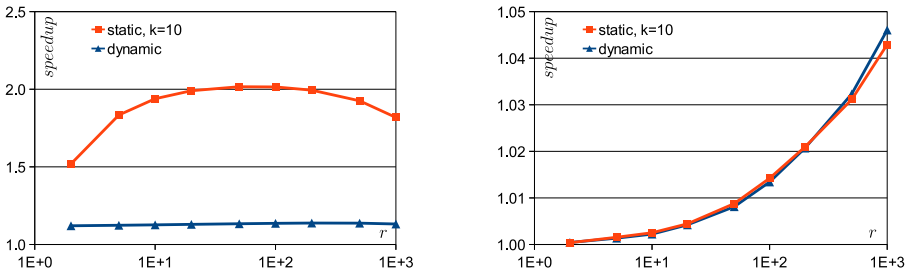
The algorithms performance will be measured by the speedup of the whole MapReduce application with load balancing (dynamic or static) in comparison to executing MapReduce as described in Section 2. Each point on the charts presents the average value over 10 instances.

We will first study the number of processors in the system executing MapReduce application. Fig. 1 presents the influence of the mapper number m on the performance of a system with $r = 100$ reducers. It can be seen that the speedup improves with growing m for both algorithms and both instance sets. This can be explained by the fact that when the number of mappers grows, then the time of mapping decreases.



a) b)

Fig. 1 Speedup vs. m. a) Hard instances, b) easy instances.



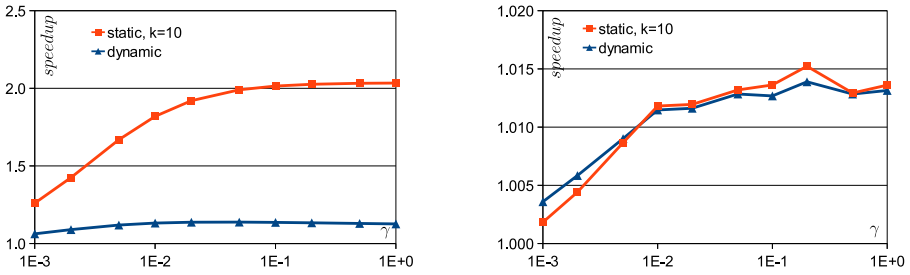
a) b)

Fig. 2 Speedup vs. r. a) Hard instances, b) easy instances.

Therefore, changes in the reducing time, which are caused by the load balancing algorithms, have a greater impact on the whole schedule length. The speedup is much larger for hard instances than for the easy ones. This is natural, as in the easy instances the initial load distribution is already balanced. Therefore, there is less room for improvement.

For hard instances the static algorithm performs much better than the dynamic method, because balancing the load dynamically can only start after all reducers finish sorting. If the load distribution is unbalanced, many processors have to wait idle until the last reducer sorts all its load. The situation is different for easy instances, for which the dynamic algorithm performs better for small m. In this case, the amounts of load sent from a mapper to a reducer are big. In consequence, the time between the beginning of computations on the first and on the last reducer is long. Therefore, it is better to create unbalanced load distribution, such that the reducers which start processing earlier obtain more data. This can be done by the dynamic method, but not by the static algorithm. When m gets larger, the reducers start computing around the same time and this effect disappears.

The influence of the reducer number r on the application speedup is shown in Fig. 2. In the instances with $r > 100$ the bisection width limit was increased to $l = r$, so that



a) b)

Fig. 3 Speedup vs. γ. a) Hard instances, b) easy instances.

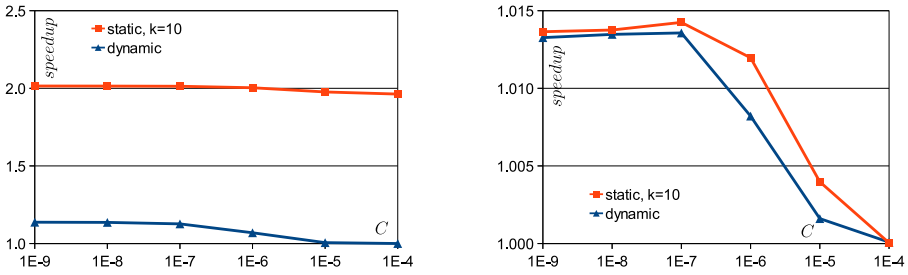
it was not limiting the communication speed. For the hard instances, the performance of both algorithms first increases, and then decreases. When the number of reducers is very small, the load distribution found by the heuristic in the static algorithm is often not very balanced. Hence, the speedup is smaller than for larger reducer numbers. When there are many reducers ($r > 100$), the load obtained by a single reducer, and hence the reducing time, decreases. Thus, we observe decreasing speedup, because the contribution of reducing to the total application running time is smaller. A similar effect can be seen for the dynamic algorithm, but its range is not so significant.

The speedup of both algorithms does not decrease for the easy instances. As the number of keys in the input data is constant, for larger r the number of keys per reducer is smaller. Hence, the load distribution between the reducers tends to be more unbalanced, and hence, the algorithms have better opportunity to fix it. This effect is stronger than the impact of decreasing the reducing time. For hard instances it is not so significant because the load distribution for small r is already skewed.

In Fig. 3 we present the speedup for different values of the multiplicity γ of results created by the mappers. As could be expected, increasing γ leads to better speedup for both algorithms and both instance sets. When γ is larger, the reducers receive more load and the significance of the reducing time, which is shortened by the algorithms, increases.

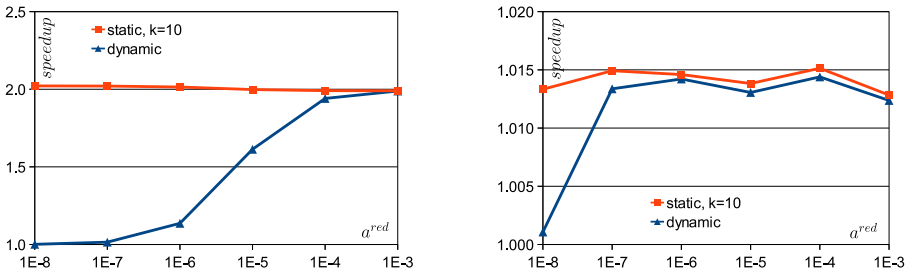
The results of experiments with changing communication rate C are presented in Fig. 4. Note that increasing C 10-fold is roughly equivalent to decreasing l also 10-fold at fixed C. Thus, Fig. 4 demonstrates also the effect of changing l on the performance. When C grows, the speedup for both algorithms decreases. This effect is stronger for the dynamic method, because it performs additional transfers of large amounts of data between the reducers. When C becomes very big, moving load between the reducers is unprofitable and the speedup achieved by the dynamic method decreases to 1. In the static method the only additional communication in comparison to MapReduce without skew handling is sending some information from the mappers to the master and from the master to the reducers. These messages are short and do not contribute much to the application running time. Hence, the results obtained by this method for the hard instances and big C are still good.

The main disadvantage of the dynamic algorithm is that it starts working after the reducers finish sorting data. If sorting dominates in the reducing time, this may



a) b)

Fig. 4 Speedup vs. C. a) Hard instances, b) easy instances.



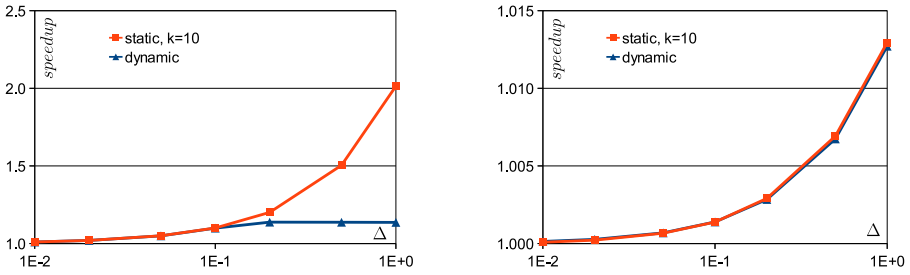
a) b)

Fig. 5 Speedup vs. a^{red}. a) Hard instances, b) easy instances.

be too late to significantly improve the execution time of the application. In Fig. 5 we present the application speedup for different values of a^{red}. Indeed, the performance of dynamic load balancing improves with growing a^{red} and reaches the same level as the static algorithm, because sorting time becomes less important. However, for the tested parameter range the dynamic method does not outperform the static one.

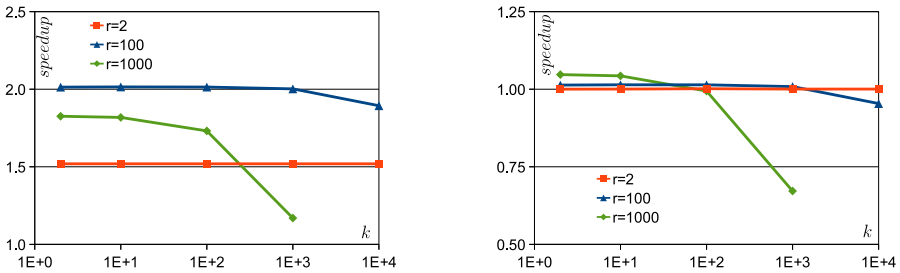
In the next set of experiments we expose the impact of parameter Δ (cf. Fig. 6). Increasing Δ makes the key frequencies more diversified and the load distribution more distorted. Therefore, both algorithms have more opportunities for introducing improvements and achieve better speedup.

Finally, let us analyze the influence of parameter k on the performance of the static algorithm. On the one hand, big k gives us greater freedom in choosing the load sizes assigned to the reducers. On the other hand, too large value of k means creating many files, sending a lot of information from the mappers to the master and from the master to the reducers, and longer master computation time. The results of the experiments with changing k, for different values of r, are presented in Fig. 7. As before, for $r > 100$ we set $l = r$ in order to avoid influencing the speedup by the communication speed. The lines for $r = 1000$ end with $k =$ 1E3, because for $k =$ 1E4 there would be more partitions than keys in the input data. It can be seen that the speedup for $k = 2$ is



a) b)

Fig. 6 Speedup vs. Δ. a) Hard instances, b) easy instances.



a) b)

Fig. 7 Speedup for the static algorithm vs. k. a) Hard instances, b) easy instances.

almost the same as for $k = 10$ (the numerical results show that $k = 10$ is slightly better for $r = 2, 100$). Thus, big values of k are not necessary to obtain good speedup. The effect of increasing k depends on the number of reducers r. When $r = 2$, the number kr of partitions created by the static algorithm is not very big even for k =1E4. The time needed for sending the information about the partition sizes to the master and for assigning the partitions to the reducers is short, and hence, the application speedup does not change. When the number of reducers grows, the impact of increasing the number of partitions per reducer becomes more visible. For the easy instances the cost of sending additional messages and assigning the partitions to the reducers is greater than the profit from load balancing.

5 Conclusions

In this work we analyzed the partitioning skew in MapReduce applications. A mathematical model of MapReduce based on divisible load theory was presented. Two methods of mitigating the skew were proposed. Both of them are easy to implement in practice and do not rely on any specific assumptions about the executed application. The static algorithm creates many data partitions on the mappers and then assigns

them to the reducers so that the load distribution becomes balanced. The dynamic algorithm transfers parts of data between the reducers, but only after sorting the intermediate data.

Both algorithms were tested in a series of computational experiments. It was shown that mitigating the partitioning skew is an important issue, as for many cases it allows for decreasing the running time of MapReduce more than twice. For most of the test instances the static algorithm performed better than the dynamic method. The disadvantage of the dynamic algorithm is that it has to wait with load balancing until all reducers finish sorting. If the time needed for reducing is not greater than the time of sorting, then sorting unbalanced data precludes obtaining good application execution time. However, there are situations in which the dynamic algorithm achieves significantly better results. Namely, for a system with a particular combination of parameters it may be necessary to divide the load between the reducers unequally in order to achieve the best performance. The dynamic algorithm can handle this situation, because it refers to the running time of the reducers, whereas the static algorithm operates on data sizes and therefore cannot construct good load partitioning in this case.

We also showed that the influence of skew mitigating depends on a combination of the system parameters. The effects of load balancing are most significant when the time of reducing dominates in the total running time of the application.

Acknowledgements This research has been partially supported by Polish National Science Centre grant N N519 643340.

References

1. R. Agrawal, H.V. Jagadish, Partitioning Techniques for Large-Grained Parallelism, IEEE Transactions on Computers 37, 1627-1634 (1988)
2. O. Beaumont, H. Casanova, A. Legrand, Y. Robert, Y. Yang, Scheduling divisible loads on star and tree networks: results and open problems, IEEE Transactions on Parallel and Distributed Systems 16, 207-218 (2005)
3. J. Berlińska, M. Drozdowski, M. Lawenda, Experimental study of scheduling with memory constraints using hybrid methods, Journal of Computational and Applied Mathematics 232, 638-654 (2009)
4. J. Berlińska, M. Drozdowski, Dominance Properties for Divisible MapReduce Computations Research Report RA-09/09, Institute of Computing Science, Poznań University of Technology (2009). `http://www.cs.put.poznan.pl/mdrozdowski/rapIIn/ra0909.pdf`
5. J. Berlińska, M. Drozdowski, Heuristics for multi-round divisible loads scheduling with limited memory, Parallel Computing 36, 199-211 (2010)
6. J. Berlińska, M. Drozdowski, Scheduling Divisible MapReduce Computations, Journal of Parallel and Distributed Computing 71, 450-459 (2011)
7. V.Bharadwaj, D.Ghose, V.Mani, T.Robertazzi, Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press, Los Alamitos, CA (1996)
8. J. Błażewicz, M. Drozdowski, Scheduling divisible jobs on hypercubes, Parallel Computing 21, 1945-1956 (1995)
9. Y.-C.Cheng, T.G.Robertazzi, Distributed computation with communication delay, IEEE Transactions on Aerospace and Electronic Systems 24, 700-712 (1988)
10. J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004)
11. M. Drozdowski, W. Głazek, Scheduling divisible loads in a three-dimensional mesh of processors, Parallel Computing 25, 381-404 (1999)
12. M.Drozdowski, Scheduling for Parallel Processing. Springer (2009)

13. B. Gufler, N. Augsten, A. Reiser, A. Kemper, Handling Data Skew in MapReduce, CLOSER '11: Proceedings of the 1st International Conference on Cloud Computing and Services Science, 574-583 (2011)
14. S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, L. Qi, LEEN: Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud, CloudCom '10: Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science, 17-24 (2010)
15. Y. Kwon, M. Balazinska, B. Howe, J. Rolia, SkewTune: Mitigating Skew in MapReduce Applications, SIGMOD '12: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, 25-36 (2012)
16. 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)
17. J. Lin, C. Dyer, Data-Intensive Text Processing with MapReduce. Morgan & Claypool (2010)
18. T.Robertazzi, Ten reasons to use divisible load theory, IEEE Computer 36, 63-68 (2003)

MISTA 2013

RCPSP with Resource Transfers and Money Management

First Author •: Alain QUILLIOT, Labex IMOBS3/LIMOS, CNRS/Blaise Pascal University

Second Author: Hélène TOUSSAINT, Labex IMOBS3/LIMOS, CNRS, FRANCE

Abstract This paper deals with *Resource Constrained Project Scheduling* (RCPSP) problems, which involve resource transfers and a *Money* resource. We use Flow models in order to formalize those extended RCPSP, derive structural results, as well as generic greedy and local search algorithms, based upon the use of an *Insertion* operator. We perform numerical tests, and end while studying the impact of ad hoc *Invest/Borrow* strategies on the *makespan* value.

1 Introduction

Dealing with *Resource Constrained Project Scheduling Problems* (RCPSP: see [1, 2, 3]) means scheduling tasks, submitted to temporal and resource constraints, while minimizing the induced *Makespan* value. This problem has been extensively studied: [4, 5, 6, 7]; its theoretical analysis requires the use of sophisticated mathematical tools: linear programming, posets, hypergraphs..: [8]. While Standard RCPSP only involves deterministic non pre-emptive tasks and renewable resources, extended models adress pre-emption: [9], time lags: [10]), non renewable resources: [11, 12], non constant profiles, robustness: [13], deadlines and penalties, redundant resources…: [14, 15, 16]. A survey about RCPSP variants is available in [17].

RCPSP problems are usually NP-Complete, and getting exact results becomes hard as soon as there are more than 60 tasks and 4 resources: [18, 19]. Characterizing benchmark computational complexity is also difficult: [20]. Exact methods are most often branch and bound, cut generation and constraint propagation based: [21, 22, 12, 23, 24, 32]. Powerful

Alain QUILLIOT
CNRS FRANCE, LIMOS UMR 6158
E-mail: alain.quilliot@isima.fr

Hélène TOUSSAINT
CNRS FRANCE
E-mail: toussaint@isima.fr

lower bounds derive from specific LP models, energetic reasoning processes or largest paths computing: [25, 26, 32]. But efficient heuristics may be designed: greedy algorithms based on priority rules or insertion techniques: [13, 27, 28], local search methods: [29, 30]. Dynamic RCPSP is most often handled through priority rule based algorithms: [28].

This paper studies two extensions of the RCPSP Problem. The first one involves *Resource Transfer Delays* and is denoted by RCPSTDP: resources are transmitted from one task to another, and those communication tasks involve delays. Such a model corresponds to the case when tasks take place on production units and when part of resources (workers, equipment…) is transported from one unit to another. It is pretty different of RCPSP with Time Lags models: [11, 17], in the sense that transfer delays are conditional to effective resource transfers between the tasks. Coping with this model requires the existence of an explicit representation of the way resources transit from one production unit to another and, thus, leads us to make appear a *Network Flow* component into our model. The second one involves a specific renewable resource, called *Money Resource*, and is denoted by RCPSMRP, and what we check that is that it is possible to cast this extended RCPSP may be cast into the same *Network Flow* framework, and handled through *Invest/Borrow* strategies (2 th RCPSMRP). Though scheduling problems involving financial constraints and objectives have already been studied: [7, 12, 17], they most often split profit and makespan optimization. As a matter of fact, to the best of our knowledge, the model which we propose here has not been studied yet.

When it comes to the design of algorithms and software for complex decision models, part of the challenge is related to fast generic development, and this may lead to a trade-off with precision. Network Flow Theory: [31] aims at providing generic tools for the management of problems which involve the circulation of goods, people, energy,...The existence of a link between the RCPSP and Network Flow Theory has already been noticed in [27, 28, 6, 14], and been used in order to get ILP formulations, as well as some specific insertion algorithms. Still, few works have explicitly involved the generic *Network Flow* machinery. So, we first explain here the way RCPSTDP and RCPSMRP may be cast into the *Network Flow* framework and state several structural results. Next, we derive from this theoretical work a generic *insertion* mechanism, close to the insertion mechanisms which were proposed in [27, 28], use it in order to design and test greedy and local search algorithms, and perform numerical experiments. Those algorithms are fast approximation algorithms, which share common generic features and link off line and on line scheduling in a natural way. Of course it would have been possible to try other approaches, for instance exact approaches based on constraint programming or meta-heuristics schemes. But what we wanted to get was an evaluation of the feasibility of an algorithmic approach which explicitly involved the main Network Flow tools.

2. RCPSP with Transfer Delays: A Flow Framework

Preliminary notations: We denote by ← the value allocation operator: "$x \leftarrow \alpha$" means that variable x takes a value equal to α. $\boldsymbol{Q}$ is the rational number set. If τ is some partial order relation, then $\bar{\tau}$ is the relation (τ or =) and $Tr(\tau)$ is the *transitive closure* of τ. A network G with node set Z and arc set E is denoted by $G = (Z, E)$. An arc e with origin/destination nodes x and y is denoted by (x, y). A partial graph of G is the restriction of G to some subset of E.

2.1. The Non Preemptive RCPSTDP Problem

A Standard RCPSP instance $\boldsymbol{I} = (V, K, R, r, d, \ll)$ is defined by:

- a set V of non pre-emptive tasks: every task v in V is endowed with some *duration* $d_v > 0$;
- a binary no circuit *precedence* relation $\ll$, which is defined on the set V;
- a finite renewable *resource* set K: the initial amount of resource $k \in K$ is denoted by R_k; during the whole time it runs, task v requires $r_{k,v}$ amount of resource k and forbids any other task to use it. Once it is over, task v gives this resource back.;

Solving $\boldsymbol{I}$ means computing, for any $v \in V$, its starting time $T_v \geq 0$, in such a way that:

- if v and $w \in V$ are such that $v \ll w$, then $T_v + d_v \leq T_w$; (*Precedence Constraint*)
- at any time $t \geq 0$, and for any $k \in K$, $\sum_{v \in U(T,t)} r_{k,v} \leq R_k$, with $U(T, t) = \{v \in V$ such that $T_v \leq t < T_v + d_v \} \subseteq V\}$ denotes the tasks which run at time t; (*Resource Constraint*)
- the *makespan* $Makespan(T) = \mathrm{Sup}_{v \in V} (T_v + d_v)$ is the smallest possible.

An instance $\boldsymbol{I\text{-}TD} = (V, K, R, r, d, \ll, D\text{-}Lag, Depot)$ of the *Resource Constrained Project Scheduling with Transfer Delays Problem* (RCPSTDP) is defined as above, while taking into account the *Delay* function *D-Lag* whose meaning is:

- tasks of V are run at different places inside some "production" space, and resources circulate between these places. At time 0, resources are all located at a same place *Depot*, and they must be back to *Depot* when the project is over. Then the *Delay* $\boldsymbol{Q}$-valued function *D-Lag*, associates, with any pair v, w in $V \cup \{Depot\}$, a value $D\text{-}lag(v, w) \geq 0$: if task v (or *Depot*) transmits some resource to task w (or to *Depot*) or if $v << w$ (v transmits some output to w), then transferring this resource requires a $D\text{-}lag(v, w)$ delay between the *ending time* of v (0 if $v = Depot$) and the *starting time of w.* (*Delay Constraint*)

So, solving $\boldsymbol{I\text{-}TD}$ means simultaneously computing a Time vector T as in the simple case, together with an ad hoc description F of the way resources are provided to the tasks.

Remark 1. RCPSTD and RCPSP with *Time Lags* are quite different problems, since delays D-lag(v, w) only impact tasks v, w which exchange resources.

2.2. Linking Network Flows with RCPSTDP: *Timed Flows*

We now propose "ad hoc description F of the way resources are provided to the tasks".

Recall: Network Flows. Given a network $G = (Z, E)$, together with a $\boldsymbol{Q}$-valued function ϕ defined on the node set Z; a $\boldsymbol{Q}$-valued E-indexed vector f is a *ϕ-flow vector* iff:

$$\Sigma_{\, e \text{ such that origin}(e) = z} f_e = \Sigma_{\, e \text{ such that destination}(e) = z} f_e = \phi(z) \quad (Kirshoff\ Law)$$

If I is some *commodity* set, if $\phi = (\phi(i), i \in I)$ is a *commodity function*, i.e., if every $\phi(i)$, $i \in I$, is a $\boldsymbol{Q}$-valued function defined on the node set Z, then we call *ϕ-flow vector* any collection $f = (f(i), i \in I)$, where every $f(i)$, $i \in I$, is a $\phi(i)$-*flow vector*.

The Activity Network. Let $\boldsymbol{I\text{-}TD} = (V, K, R, r, d, \ll$, D-Lag, *Depot*) be a RCPSPTD instance. We derive from $\boldsymbol{I\text{-}TD}$ the *Activity Network* $\boldsymbol{N}(V) = (V^*, E^*)$ by introducing two auxiliary tasks *Start* and *End*, and by setting:

- $V^* = V \cup \{Start, End\}$ = node set of $\boldsymbol{N}(V)$;
- $E^* = \{(v, v'), v, v' \in V\} \cup \{(Start, v), v \in V \cup \{End\}\} \cup \{(v, End), v \in V \cup \{Start\}\}$.

We define the E^*-indexed *length vector* d^* by setting:

- for any $v \in V$, $d^*_{(Start, v)} = D\text{-}Lag(Depot, v)$ and $d^*_{(v, End)} = d_v + D\text{-}Lag(v, Depot)$;
- $d^*_{(End, Start)} = -\infty$;
- for any $v \in V$, $w \in V \cup \{End\}$, $d^*_{(v, w)} = d_v + D\text{-}Lag(v, w)$.

We provide the node set V^* with a *commodity vector* r^*, by setting, for every $k \in K$ and for any v in V^*: if $v \in V$ then $r^*_{k,v} = r_{k,v}$ else $r^*_{k,v} = R_k$. We define the *precedence arc subset* $E^*_{\ll}$ by setting: $E^*_{\ll} = \{(v, v'), v, v' \in V$ such that $v\ Tr\ (\ll)\ v'\} \cup \{(Start, v), (v, End), v \in V\}$.

Feasible solutions of *I-TD* and r^*-flow vectors. Then, an explicit representation of the circulation of resources between the tasks of V, consists into a r^*-flow vector $F = (F(k), k \in K)$, defined on the *Activity Network* $\mathbf{N}(V)$, which may be viewed as transporting resources $k \in K$, from *Depot* (source-node *Start*) to *Depot* (end-node *End*), while providing the activities $v \in V$ with the required resources. The *support arc subset* $E(F, \ll)$ of F is the set $E^*_{\ll} \cup \{(v, w), v, w \in V$ such that $F_{(v, w)}$ is non null$\}$. Clearly, $E(F, \ll)$ is *no circuit*. In fact, even if we only deal with a simple RCPSP instance $\boldsymbol{I} = (V, K, R, r, d, \ll, D\text{-}Lag = 0)$, we may derive, as described in Figure 1, such a flow F from any feasible schedule T.

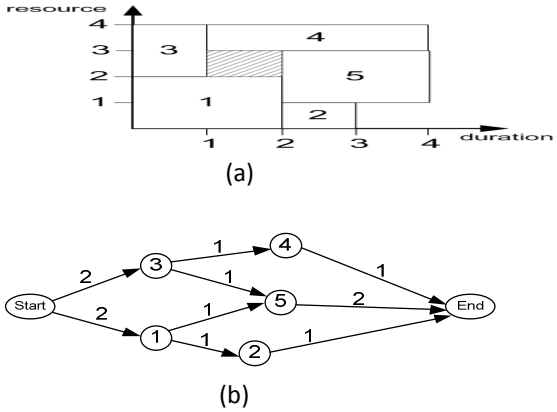


Figure 1: (a) Gantt chart – (b) Flow representation

Casting RCPSPTDP into a formal framework: *Timed Flows.* Let F be a no circuit r^*-flow vector defined on the *Activity Network* $\mathbf{N}(V)$ and T be a time V-indexed vector such that (F, T) is a feasible solution of ***I-TD***. If we extend the time vector T to $V \cup \{Start, End\}$ by setting:

- $T_{Start} = 0$; $T_{End} = \Delta = Makespan(T) = \mathrm{Sup}_{v \in V}\,(T_v + d_v + D\text{-}Lag(v, Depot))$;

Then, for any arc $e = (v, w)$ in the arc set E^* of $\mathbf{N}(V)$, we get the implication:

$$e = (v, v') \in E(F, \ll) \Longrightarrow (T_w \geq T_v + d_v \Leftrightarrow T_w \geq T_v + d^*_e) \qquad \text{(P1)}$$

This leads us to define a *Timed* (r^*, d^*)*-Flow* as being any such pair (F, T) made of a no circuit r^*-flow vector F and a time vector T such that (P1) is true. One easily checks that any such a *Timed* (r^*, d^*)*-Flow* (F, T) defines a feasible solution of ***I-TD***. This notion of *Timed Flow* allows us to reformulate RCPSTDP as follows:

RCPSPTD Timed Flow Reformulation: *solving the RCPSTDP instance* ***I-TD*** *= (V, K, R, r, d,* $\ll$*, D-Lag, Depot) means computing, on the Activity network* $\mathbf{N}(V)$*, a Timed* (r^*, d^*)*-Flow (F, T) such that* T_{End} *is the smallest possible.*

Also, following ([27, 28]) one easily checks that:

Theorem 1: Standard RCPSP Reformulation Theorem
Any feasible solution T of the Standard RCPSP instance ***I*** *= (V, K, R, r, d, ≪) may be extended into a feasible solution (F, T) of the RCPSTDP instance* ***I-TD*** *= (V, K, R, r, d, ≪, 0, Depot).*

2.3. Connectivity Theorem

Part of the efficiency of the flow machinery derives from properties of the flow polyhedron.

The No Circuit r^*-Flow Polyhedral Vertex Set: the set of all r^*-flow vectors $F \geq 0$ defined on the *Activity Network* $\boldsymbol{N}(V)$ defines a bounded polyhedron P_{r^*}. A r^*-flow vector F is a *vertex* of P_{r^*} if it contains no *non null alternated cycle*, that means no cycle $(v_0, v_1, ..., v_n = v_0)$ such that:

- n is even and all the nodes $v_0, v_1, ..., v_{n-1}$ are distinct (the cycle is elementary);
- there exists $k \in K$ such that:
  - the arcs $(v_0, v_1), (v_2, v_3), ..., (v_{n-2}, v_{n-1})$ are all endowed with non null $F(k)$ values;
  - the arcs $(v_2, v_1), (v_4, v_3), ..., (v_0, v_{n-1})$ are all endowed with non null $F(k)$ values.

We denote by S_{r^*} the vertex set of this polyhedron. This vertex set is endowed with a canonical *adjacency* relation R: F, F' in S_{r^*} are R-adjacent if there exists some resource $k_0 \in K$, some alternated cycle Γ and some number $\lambda \geq 0$, such that, for any $k \neq k_0$, $F(k) - F'(k) = 0$; $F'(k_0) - F(k_0) = \lambda . f^{\Gamma}$, being elementary flow vector which derives from the cycle Γ; In such a case, λ is unique, and F' derives from F through *redirection of F(k) on* Γ.

It comes from LP Theory that S_{r^*} is connected for the relation R. Since we deal here with no circuit r^*-flows, we may ask about the set SN_{r^*} of the vertices of P_{r^*} which define no circuit r^*-flow vectors. We call SN_{r^*} the *No Circuit r^*-Flow Polyhedral Vertex Set.* Then we may state:

Theorem 2: Connectivity Theorem
If we suppose that, for any $k \in K$, $v, w \in V$, *we have:* $r_{k,v} + r_{k,w} \leq R_k$ *(Parallelism Hypothesis), then the set* SN_{r^*} *is connected for the adjacency relation R.*

Comment: the meaning of this result is that one may handle timed flows (and RCPSTDP instances) through classical local search procedures (cancelling cycle procedures…).

Proof of theorem 2 (*Sketch of the Proof***)**
Let us say that a no circuit r^*-flow vector $F \geq 0$ is linear if it is such that the transitive extension of the support arc set $E(F, \ll)$ is linear. So we denote by SNL_{r^*} the subset of SN_{r^*} which is made of linear r^*-flow vectors. If σ is some linear ordering of $V \cup \{Start, End\}$, which is compatible with $\ll$, we denote by $SN_{r^*}(\sigma)$ the subset of SN_{r^*} which corresponds to the case when σ may be viewed as a linear extension of the transitive extension of $E(F, \ll)$. Then we check, by using ad hoc flow redirection processes involving cycles with length 4, that:

Lemma. *$SN_{r^*}(\sigma)$ is connected for the R relation. Also, if we suppose that, for any $v \in V$, $k \in K$, $r_{k,v} \neq 0$, and that the parallelism holds, then we get that:*

- *If F and F' are both in the set SNL_{r^*}, then there exists a R-path from F to F';* (P2)
- *for any linear ordering σ of $V \cup \{Start, End\}$, which is compatible with $\ll$, the intersection of SNL_{r^*} and $SN_{r^*}(\sigma)$ is non empty.* (P3)

Clearly, this lemma allows us to conclude to the *R*-connectivity of SN_{r*} when, for every *v* in *V*, *k* in *K*, $r_{k,v}$ is non null. In order to get the general case, we use a topology trick. Let $\delta > 0$ be a small positive number. For every *v* and any *k*, such that $r_{k,v} = 0$, we replace $r_{k,v}$ by δ, and R_k by $R_k + \mathrm{Card}(V(k)).\delta$, where $V(k) = \{v \in V$ such that $r_{k,v} = 0\}$. We denote by $S_{r*}{}^{\delta}$ and $SN_{r*}{}^{\delta}$ the respective related vertex sets and by R^{δ} the related adjacency relation. $SN_{r*}{}^{\delta}$ is connected for R^{δ}. Also, if *F* is some vertex in SN_{r*}, then the *r**-flow vector F^{δ} defined by:

- for any *v* and any *k* such that $r_{k,v} = 0$, $F^{\delta}(k)_{(Start,\ v)} = \delta = F^{\delta}(k)_{(v,\ End)}$;
- $F^{\delta}(k)_{(Start,\ s)} = \mathrm{Card}(V(k)).\delta$;
- for any other pair (*e*, *k*), *k* in *K*, *e* in the arc set of the network $\boldsymbol{N}(V)$, $F^{\delta}(k)_e = F(k)_e$;

is no circuit and does not admit any non null alternated cycle, and thus is in $SN_{r*}{}^{\delta}$. Then we check that any pair *F*, *H* in SN_{r*}, may be connected by a path Γ which is the limit, when δ converges to 0, of paths Γ^{δ}, $\delta > 0$, where every path Γ^{δ} connects F^{δ} and H^{δ} in $\mathrm{SN}_{r*}{}^{\delta}$. *End-Proof.*

Remark 2: one easily checks that the *Parallelism* hypothesis cannot be removed.

3. Insertion Operators and Algorithms.

As told in 2, introducing the *Timed Flow* formalism is motivated by the prospect of applying flow algorithmic tools to RCPSTDP instances. More specifically, what we shall do in current Section 3 is translating previous *Connectivity* Theorem 2 into generic algorithmic devices. This approach reflects the main goal of our work, which aims at deriving from the Network Flow Framework efficient approximation algorithms, with generic features which make them easy to implement and adapt to dynamic contexts or additional constraints. Clearly, other approaches would be possible: we could for instance try to apply the constraint programming paradigm to a formulation of our problem which would involve cumulative constraints (see [32]). Our feeling is that, in any case, we should find a way to simultaneously deal with resource routing and task scheduling, and that this would lead us to higher implementation costs while still restricting ourselves to the search of approximate solutions.

Basically, our RCPSTDP algorithms works while performing *insertion*/*removal* processes close to those which have been proposed in [27, 28] for standard RCPSP: the basic difference lays upon the fact that every time such an insertion/removal process is performed, it involves the exact resolution of a specific *Insertion-Flow* sub-problem related to a given *Cut* of the currently inserted task set: the related resolution process updates all flow values which transit between both sides of this *Cut*, and implements the *Connectivity Theorem* of Section II. More precisely, at any time during the process of some RCPSTDP instance ***I-TD*** = (*V*, *K*, *R*, *r*, *d*, $\ll$, *D-Lag*, *Depot*), we are provided with some *Inserted Activity* subset *W* of *V*, with a no circuit r^*-flow vector *F* defined on the *Activity Network* $\boldsymbol{N}(W)$, and with two positive (or null) $\boldsymbol{Q}$-valued time vectors *T* and T^*, both with indexation on W^*, in such a way that, for any *v* in W^*: (P4)

- T_v = Length of a *d**-largest path from *Start* to *v* in the *Partial Network* defined by $E(F, \ll)$;
- T^*_v = Length of a *d**-largest path from *v* to *End* in the *Partial Network* defined by $E(F, \ll)$.

Clearly, (*F*, *T*) defines a timed (r^*, d^*)-flow on $\boldsymbol{N}(W)$. Then, performing an *Insertion* means picking up some task v_0 which is not in *W*, computing some *Cut*, i.e. a partition of *W* into two subsets *U* and $W-U$, such that no flow amount goes from to $(W-U) \cup \{End\}$ to $U \cup \{Start\}$, and turning (*F*, *T*), while solving some *Insertion-Flow* Problem, into a convenient timed (r^*,

d^*)-flow defined on $N(W \cup \{v_0\})$, in such a way that v_0 receives flow from $U \cup \{Start\}$ and gives it back to $(W-U) \cup \{End\}$. Performing a *Removal* means reversing this operation. In order to better explain those mechanisms, we shall introduce the *Insertion-Flow Problem*. Meanwhile, we illustrate the general insertion/removal mechanism through the drawings of Figure 1, which represent, (a), a partial solution with 4 tasks and a *Cut* ($U = \{1, 3\}$, $W-U = \{2, 4\}$), and, (b), the insertion of task $v_0 = 5$ (with $d_5 = 2$ and $r_5 = 2$) into this *Cut*.

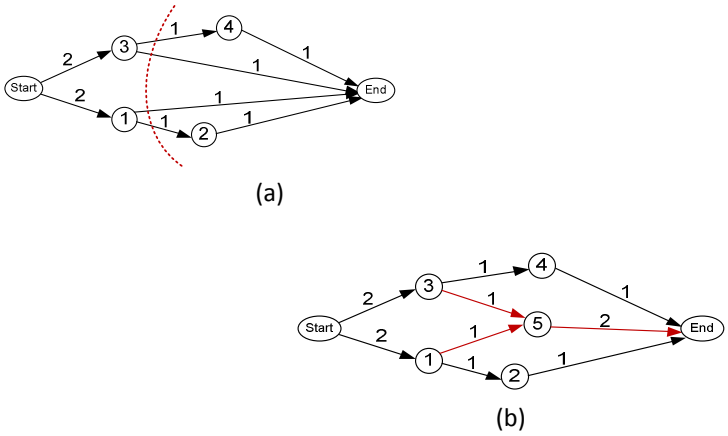


Figure 2: (a) The flow and the "cut" (dotted line) - (b) the resulting flow after insertion

3.1. The Insertion Flow Problem

We formalize the *Insertion-Flow Problem* related to a given *Cut* as follows: we say that an oriented graph $N = (X, E)$ is *almost-bipartite*, if there exists some node z_0 in X such that the restriction of N to $X - \{z_0\}$ is bipartite, which means that $X - \{z_0\}$ may be written as the disjoint union $X - \{z_0\} = A \cup B$, of two disjoint independent sets A and B; let us suppose now that we are endowed with two positive (or null) $\mathbf{Q}$-valued A-indexed vectors Π, Out, with two positive (or null) $\mathbf{Q}$-valued B-indexed vectors Π^*, In, with some $\mathbf{Q}$-valued positive vectors Δ with indexation on $(A \cup \{z_0\}).(B \cup \{z_0\})$, and with a positive (or null) coefficient ρ, such that: $\Sigma_{x \in A}\, Out_x = \Sigma_{y \in B}\, In_y \geq \rho$; then we say that a vector $G = (G_{x,y} \geq 0, x \in A \cup \{z_0\}, B \cup \{z_0\}) \geq 0$, is an *Insertion-Flow vector* related to (X, E), z_0, A, B, In, Out, Π, Π^*, Δ, ρ) iff:

- for any x in A, $Out_x = \Sigma_{y \in B \cup \{x0\}}\, G_{x,y}$; for any y in B, $In_y = \Sigma_{x \in A \cup \{x0\}}\, G_{x,y}$;
- $\rho = \Sigma_{y \in B}\, G_{z0,y} = \Sigma_{x \in A}\, G_{x,z0}$.

For such an *Insertion-Flow* vector G, we set:

- $Make1(G) = \mathrm{Sup}_{x \in A,\, y \in B \text{ such that } (x,y) \in E \text{ or } Gx,y \neq 0}\, (\Pi_x + \Pi^*_y + \Delta_{x,y})$;
- $Make2(G) = \mathrm{Sup}_{x \in A,\, y \in B \text{ such that } ((x,z0) \in E \text{ or } Gx,z0 \neq 0) \text{ and } (z0,y) \in E \text{ or } Gz0,y \neq 0)}\, (\Pi_x + \Pi^*_y + \Delta_{x,z0} + \Delta_{z0,y})$;
- $I\text{-}Makespan(G) = \mathrm{Sup}(Make1(G), Make2(G))$.

This definition leads us to introduce the following ***Insertion-Flow Problem***: *{Given (X, E), z_0, A, B, In, Out, Π, Π^*, Δ, ρ, as above. Find a related insertion flow vector G in such a way that I-Makespan(G) be the smallest possible.}*

Explanation: if we refer to the global process with Card(K) = 1, then we should think: (P5)

- $A = U \cup \{Start\}$; $B = (W - U) \cup \{End\}$; $E = (v, w)$, $v \in A$, $w \in B$, such that $v\ Tr(\ll)\ w$;
- $z_0 = v_0$; $\rho = r_{k,v0}$;
- for any v in $U \cup \{Start\}$, $\Pi_v = T_v$ and $Out_v = \Sigma_{w \in B}\, F(k)_{(v,w)}$;

- for any w in $B = (W - U) \cup \{End\}$, $\Pi^*_w = T^*_w$ and $\underline{In}_w = \Sigma_{v \in A} F(k)_{(v,w)}$.
- for any pair v, w in $(A \cup z_0).(B \cup z_0)$, $\Delta_{v,w} = d^*_{(v,w)}$.

We deal with this *Insertion-Flow* problem, through the following *Insertion-Flow* procedure:

***Insertion-Flow* Procedure**

For any *Insertion-Flow* vector G, we set: $u(G)) = \text{Sup}\ (\Pi_x + \Delta_{x,z0})$, $x \in A$ such that ($G_{x, z0} \neq 0$ or $(x, z_0) \in E$); Then compute the set $\Lambda_{E, z0}$ of the possible *attachment values* $u(G)$);

For any u in $\Lambda_{E, z0}$, **do**
Compute some *Insertion-Flow* G such that $u = u(G)$;
While *Possible* **do** Make decrease, by searching some *Redirection Path*, the G value on the arcs (x, y) such that *I-Makespan*$(G) = G_{x,y}$, while keeping $u(G)$ unmodified;
Keep the best *Insertion-Flow* G which was computed.

Theorem 3: Insertion-Flow Theorem
Insertion-Flow solves, in an exact way, the Insertion-flow Problem in polynomial time.

Proof of theorem 3 (*Sketch of the Proof*): The key point is that $u(G)$ remains unmodified during the *While* loop: then, a standard flow reasoning makes appear that if G is some *Insertion-Flow* vector, if $G(u)$ derives from the *While* loop for a given u, and if we have both $u(G) = u$ and *I-Makespan*$(G(u)) >$ *I-Makespan*(G), then the cycle decomposition of the flow vector $G - G(u)$ makes appear some *Redirection Path* Γ which improves $G(u)$. *End-Proof.*

***Fast-Try-Insertion* and *Lex-Insertion-Flow* Procedures**: As a matter of fact, before insertion is effectively performed, it must be tried several times. In order to speed this process, we use a greedy *Fast-Try-Insertion* heuristics, which ties the node z_0 with the nodes x of A which are the most likely to make increase *I-Makespan* in case $G_{x, z0} = 0$. Also we minimize the number of arcs (v, w) such that (v (Not $Tr(<<)$) w and $F_{(v, w)} \neq 0$) by turning *Insertion-Flow* into a *Lex-Insertion-Flow* procedure which works by concentrating G values on the same arcs of E.

3.2. Generic Flow Algorithms for the RCPSTDP

A Greedy Insertion Algorithm *RCPSTDP-Greedy-Flow:* it works through successive insertions as explained at the beginning of Section III. Clearly, a key point is about the computation of the *Cut U*, prior to the call to the *Insertion-Flow* procedure. But searching for a best Cut U in the general sense is difficult. As a matter of fact, we may state:

Theorem 4: Cut Theorem: *The search for a best Cut as defined above is NP-Complete*.

Proof of theorem 4: Let us consider the following input situation for the *Best Cut* Problem:
- there is only one resource (so we set $r_{k,x} = r_x$) and $R_k = R$, for the unique k in K;
- $\sum_{v \in W} r_v = R$; $r_{v0} = R/2$; $\ll$ is the empty relation; for any v in W, $d_v = 1$; $d_{v0} = 2$.

Determining whether the optimal value of this *Best Cut* instance is no more than 2 means solving some *2-Partition* problem instance. *End- Proof.*

Still, if v is some node $W \cup \{End\}$, we may set $Cut(v) = \{w \in W, w \neq v$, such that $T_w \leq T_v\}$. So, while searching for a best Cut U is difficult, we can easily scan the set W, and choose $U = Cut(v)$ in such a way that an application of the *Insertion-Flow* Procedure yields the best possible *Makespan* value. The whole process, which may be randomized, comes as follows:

RCPSP-Greedy-Flow Algorithm

Input: the instance ***I-TD*** = (V, K, R, r, d, $\ll$, D-Lag, *Depot*) of the RCPSTDP problem;
Output: a timed (r^*, d^*)-flow (F, T), and the related *Makespan* value Δ

Initialization: $W \leftarrow Nil$; $\Delta \leftarrow 0$; $F \leftarrow 0$; $T_{Start} = T_{End} = T^*_{Start} = T^*_{End} \leftarrow 0$;
Main Loop: ***RCPSP-Flow*** $\leftarrow$ *Packet-Insertion*(*Nil*);

Packet-Insertion **Procedure**

Input: the instance ***I-TD*** = (V, K, R, r, d, $\ll$, D-Lag, *Depot*), a subset W of V, given together with a related no circuit r^*-flow vector F, and two vectors T and T^* as in (P4).
Output: a timed (r^*, d^*)-flow (F, T), and the related *Makespan* value Δ
While $W \neq V$ **do**
Randomly Pick up v_0 in $V - W$;
Compute v_1 in $W \cup \{End\}$, such applying *Fast-Try-Insertion* to $Cut(v_1)$ yields the best possible *Makespan* value, together with a related attachment value u_1 ;
Apply *Lex-Insertion-Flow* to $Cut(v_1)$ and u_1, and insert v_0 into (F, T); $W \leftarrow W \cup \{v_0\}$;

A local search *RCPSTDP-LS-Flow* Algorithm

The above *Packet-Insertion* operator gives rise in a generic way to a local search operator *Transform-Insertion*. The idea is that, once we are provoded with a timed (r^*, d^*)-flow (F, T) defined on the activity set V, we may pick up some (small) subset S of V, take it away from V (which means *reversing* the insertion process, i.e applying some *Reverse-Insertion* procedure) and, next, come back to inserting the activities of S into the pair (F, T):

Transform-Insertion*(S)* **Procedure*:*** *Packet-Reverse-Insertion*(S); *Packet-Insertion*($V - S$).

Provided with this operator, we may design local search algorithms involving *Descent*, *Tabu* or *Simulated Annealing*, while picking up S as the task subset defined by a critical path (*Crit-Path* strategy) or by the tasks which are running at some critical time t (*Antichain* strategy).

4. Numerical Experiments on RCPSTDP.

We performed several experiments, on PC AMD opteron 2.1GHz, while using gcc 4.1 compiler. We had to cope with the fact that RCPSPTD has not been previously studied, and so that no benchmarks were available for this model. As told before, we did not try to design and implement alternative resolution algorithms, since it would have represented a very significant additional computational effort. Instead, we first tested the case *D-Lag* = 0, which means RCPSPTD is the standard RCPSP, on the PSPLIB testbed. Next, we designed *ad hoc* instances, some of them derived from PSPLIB, in such a way that we knew in advance their optimal values.

4.1. Experiments on PSPLIB instances with $\mathcal{D}$-$\mathcal{L}ag$ = 0.

Tables 1 and 2 provide us with average results for ***RCPSTDP-Greedy-Flow*** and ***RCPSP-LS-Flow***, related to the PSPLIB packages: 30 jobs, 60 jobs, 120 jobs, *D-Lag* = 0, with:

- *N-Ac* = number of activities, *N-Res* = number of resources, *N-Re* = number of replications.
- *Time* = CPU time for *N-rep* replications; *Gap-LB* (%) = gap between our values and: in case of 30 job instances, the optimal value; in case of 60 and 120 job instances, the best lower bound value; *Gap-TB (%)* = gap between our values and: in case of 30 job instances, the optimal value; in case of 60 and 120 job instances, the largest path lower bound value.

In this specific case, *Fast-Try-Insertion* is used. The induced results are very satisfactory, taken into account the simplicity and the generic features of our algorithms. In case of 60 and 120 job instances, they provide us with some of the best available results.

Table 1: *RCPSP-Greedy-Flow* procedure, Mean Results

N-Ac	*N-res*	*N-re*	*Time*(*s*)	*Gap-TB*	*Gap-LB*
30	4	100	0.63	1.87	1.87
30	4	1000	6.3	0.92	0.92
60	4	100	4.54	16.91	7.10
60	4	1000	53.04	15.37	5.79
120	4	100	29.6	52.33	21.32
120	4	1000	515	48.84	18.5

Table 2: *RCPSP-LS-Flow* procedure, Mean Results

N-Ac	*N-Re*	*Strategy*	*Gap-TB*	*Gap-LB*	*Time* (*s*)
30	50	Crit-Path	0.49	0.49	19.41
30	50	Antichain	0.12	0.12	8.42
60	50	Crit-Path	13.69	4.4	101.50
60	50	Antichain	11.93	3.34	46.93
120	10	Crit-Path	45.41	15.80	210.80
120	10	Antichain	38.67	12.41	63.73

4.2. PSPLIB instances such that RCPSP and RCPSTDP optimal values are the same.

These instances are “difficult” ones, built as follows: starting from a standard RCPSP instance ***I*** and from some almost optimal solution *T* , we compute, for every pair (*v*, *w*), the maximal *D-Lag* value *Max-Lag*(*T*, *v*, *w*) which is compatible with *T*, and generate *D-Lag* with a mean ratio *D-Lag*/*Max-Lag* between 10% to 50%. Table 3 provides us with results related to 30 job instances distributed into 5 groups, according to the value *D-Lag*/*Max-Lag*, (*N-Re* = 100).

Table 3: *RCPSP-Greedy-Flow* procedure, Mean Results, Ad Hoc PSPLIB Instances

Group-Instance	*Gap* (%)	*Time* (s)
1 - 10%	2,50	2,71
2 – 20%	3,21	2,79
3 – 30%	7.35	2,92
4 – 40%	8,50	2,94
5 – 50%	14,50	3.12

Comment: Clearly, the largest is the *D-Lag*/*Max-Lag* mean ratio, the most difficult are the instances. Still, results remain satisfactory. Also, we notice that using the full *Try-Insertion* procedure has an impact on the speed of the process.

4.3. *Ad Hoc* instances such that RCPSP and RCPSTDP optimal values are the same.

These tests are related to ad hoc instances ***I*** with only one resource, which have been generated in such a way that the optimal schedule *T* of ***I*** has a value equal to $(\Sigma_{v \text{ in } V} r_v.d_v)/R$. We generate *D-Lag* values and the << relation in such a way that the ratio between the mean *D–Lag* values and the mean length of the actions of *V* varies from 10% to 50%. The following figure 3 shows the *skeleton* of such an instance. In this case, we may set 4 << 22 and a *D-Lag*(4, 22) value may be generated between 0 and *Max-Lag*(4, 22) = 7.

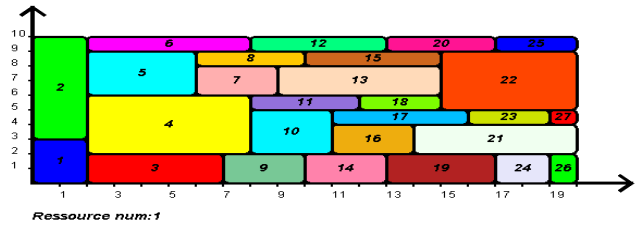


Figure 3: the *skeleton* of an Ad Hoc 1 resource instance

Following Table 4 provides us with related results, with *N-Act* between 22 and 37, and *N-Re* = 100. One notices that those specific instances are more difficult than the previous one.

Table 4: *RCPSP-Greedy-Flow*, *N-Re* = 100, Ad Hoc Instances with High Parallelism Level

Group-Instance	*Gap* (%)	*Time* (s)
1 - 10%	13,50	2,11
2 – 20%	13,00	2,11
3 – 30%	22,00	2,12
4 – 40%	18,50	2,32
5 – 50%	84,50	2,54

4.4. Fast-Try-Insertion versus Full Try-Insertion.

In this last experiment (same instances as in 4.3) , we replace *Try-Insertion* of 3.1, which involves path search, by a greedy *Fast-Try-Insertion* procedure, and test the *Gap* increase.

Table 5: *RCPSP-Greedy-Flow*, *N-Re* = 100, Mean Results, *Fast-Try-Insertion*

Group-Instance	Gap (%)	Time (s)
1 - 10%	2.85	0,71
2 – 20%	3,80	0,79
3 – 30%	8.23	0,92
4 – 40%	9.80	0,94
5 – 50%	18,10	1.12

5. RCPSP with Money Resource

In this section we introduce non renewable resources. As a matter of fact, we consider a specific kind of non renewable resource, which we call *Money* resource, which may vary in a very significant way, since it is produced by some tasks and consumed by others.

An instance **I** = (V, K, R, r, d, $<<$, Φ, ϕ, φ, δ) of the *Resource Constrained Project Scheduling with Money Resource Problem* (RCPSMRP) is defined by the same components (V, K, R, r, d, $<<$) as in the standard case, augmented with an initial *Money* amount Φ, with two *Money Resource* function ϕ and φ and with a delay function δ, whose meaning comes as follows:

- Launching a task v of V requires $\phi(v)$ *Money*, and achieving it allows the project manager getting back $\varphi(v)$ *Money*, which become available $\delta(v)$ times unit after v has been delivered. In case $\varphi(v)$ is null, $\delta(v)$ is also null;
- At time 0, the initial *Money* amount which is available is Φ;
- The process ends when all the tasks are done and related due money is collected.

Then solving the RCPSMRP instance $\boldsymbol{I}$ = (V, K, R, r, d, $<<$, Φ, ϕ, φ, δ) means computing, for any task $v \in V$, its *starting time* $T_v \geq 0$ in such a way that:

- if v and $w \in V$ are such that $v << w$, then $T_v + d_v \leq T_w$; (*Precedence Constraint*)
- at any instant $t \geq 0$, and for any resource $k \in K$, we have $\Sigma_{v \in U(T, t)} r_{k,v} \leq R_k$, where the subset $U(T, t) \subseteq V$ is defined as the set of the tasks which are under execution at time t: $U(T, t) = \{v \in V$ such that $T_v \leq t < T_v + d(v)\}$; (*Resource Constraint*)
- at any instant $t \geq 0$, the available money amount $M(T, t) = \Phi + (M^+(t)$ = Money generated by the tasks which have been achieved before time $t) - (M^-(t)$ = Money which has been spent before time $t)$ is non-negative; (*Money Constraint*)
- the *Makespan* value *F-Makespan*$(T) = \text{Sup}_{v \in V}(T_v + d_v + \delta(v))$ is the smallest possible.

Remark 3: checking the feasibility of a RCPSMRP instance $\boldsymbol{I}$ is NP-Complete. In case $\Phi = 1$ and $V = A \cup B$, with: Card(A) = Card(B); for any v in A, $\phi(v) = 1$ and $\varphi(v) = 0$; for any v in B, $\phi(v) = 0$ and $\varphi(v) = 1$; then checking the feasibility of $\boldsymbol{I}$ contains the *Bandwidth Problem on Co-Bipartite Balanced Graphs*, which is NP-Complete (see [6]).

Remark 4: As it is stated here, RCPSMRP has not been studied yet. Most RCPSP model involving cash management introduced money as a criterion which had to optimize, together with the makespan criterion (see [7, 12, 17]). It can be understood as a case of RCPSP with non-renewable resources, but the very specific feature of the money resource, that some task may produce money in a significant way, usually does not appear in non -renewable resource models, where non renewability most often derives from wearing.

5.1. Adapting the Flow Model to a RCPSMRP Instance.

Once again, our idea is to check that the flow framework is generic enough to allow a flow formulation of the RCPSMRP Problem. So, we define, on the *Task Network* $\boldsymbol{N}(V) = (V^*, E^*)$, an additional *length vector* δ^* by setting:

- for any $v \in V$, $\delta^*_{(Start, v)} = 0$; $\delta^*_{(End, Start)} = 0$;

- for any $v \in V$, $w \in V \cup \{End\}$, $\delta^*_{(v, w)} = d_v + \delta(v)$.

We also define, on the node set V^*, two *money vectors* ϕ^* and φ^*, by setting, for every v in $V^* = V \cup \{Start, End\}$:

- if $v \in V$ then $\phi^*_v = \phi(v)$ and $\varphi^*_v = \varphi(v)$;
- $\varphi^*_{Start} = \Phi$; $\phi^*_{Start} = 0$; $\varphi^*_{End} = 0$; $\phi^*_{End} = \Phi - \Sigma_{v \in V} \phi(v) + \Sigma_{v \in V} \varphi(v)$.

As in Section 2, we easily check that, with any feasible solution T of the RCPSMRP instance $\boldsymbol{I} = (V, K, R, r, d, <<, \Phi, \phi, \varphi, \delta)$, we may associate a pair $(F, F\text{-}cash)$, where:

- $F = (F(k)_{(v, w)}, k \in K, (v, w) \in E^*$ is a r^*-flow vector;
- $F\text{-}cash = (F\text{-}cash_{(v, w)}, (v, w) \in E^*$ is such that:
  - for any task v in V, $\Sigma_{v \text{ is the origin of } e} F\text{-}cash_e = \varphi(v)$; $\Sigma_{v \text{ is the extremity of } e} F\text{-}cash_e = \phi(v)$;
  - $\Sigma_{Start \text{ is the origin of } e} F\text{-}cash_e = \Phi = \varphi^*_{Start}$;
  - $\Sigma_{End \text{ is the extremity of e}} F\text{-}cash_e = \phi^*_{End} = \Phi - \Sigma_{v \in V} \phi(v) + \Sigma_{v \in V} \varphi(v)$;

  We say that $F\text{-}cash$ is a (ϕ^*, φ^*)-flow vector.

Interpretation: the vector $(F, F\text{-}cash)$ transports resources $k \in K$ and *Money* from *Start* to *End*, and provides the tasks $v \in V$ with those resources in such a way it allows them to be run.

Example: We consider 4 tasks *A*, *B*, *C*, *D*, with durations 2, 5, 3, 4, together with 1 resource. Resource requirements are given by the following table:

	A	B	C	D
R	3	1	2	4
Cash	(5, 2)	(5, 3)	(2, 8)	(1, 12)

Then we get the following solution:

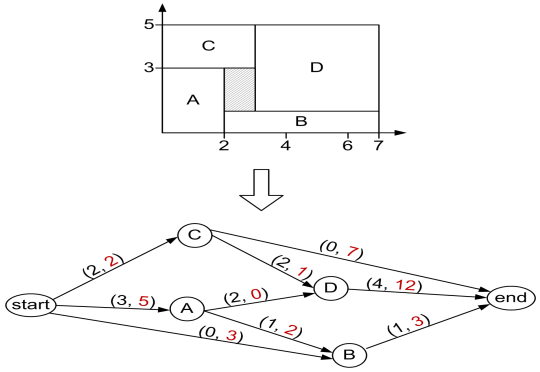


Figure 3/ A RCPSMRP Flow Solution

Explanation. The first number denotes the resource flow and the second one the *Cash* flow. Initial *Money* amount is 10, initial resource amount is 5, payment delays are null.

Provided with a vector $(F, F\text{-}cash)$ as above, we set: $E2(F\text{-}cash) = \{(v, w), v, w \in V$ such that $F\text{-}cash_{(v, v')}$ is non null$\}$, and define the *Support Arc Subset* $E^*(F, F\text{-}cash, <<)$ of $(F, F\text{-cash})$ as the *no circuit* arc set $E(F, <<) \cup E2(F\text{-}cash)$, where $E(F, <<)$ is defined as in Section 2. Then we extend T to $V \cup \{Start, End\}$ in a natural way by setting $T_{Start} = 0$ and $T_{End} = F\text{-}Makespan(T)$, we get, for any arc $e = (v, w)$ in the arc set E^*, the following implications:

- $(v, w) \in E(F, <<) \Rightarrow T_w \geq T_v + d_v \Leftrightarrow T_w \geq T_v + d^*_e$; (P6)
- $E2(F\text{-}cash) \Rightarrow T_w \geq T_v + d_v + \delta(v) \Leftrightarrow T_w \geq T_v + \delta^*_e$. (P7)

We call a triple (*F*, *F-cash*, *T*) which satisfies (P6, P7) a *Timed (r*, ϕ*, φ*)-Flow*. We check:

Theorem 5: Reformulation Theorem for the Money Case.
Solving the RCPSFRP instance **I** *= (V, K, R, r, d, <<, Φ, ϕ, φ, δ) means computing a Timed (r*, ϕ*, φ*)-Flow(F, F-cash, T) such that F-Makespan*(T) = $Sup_{v \in V} (T_v + d_v + \delta(v))$ *is the smallest possible*.

Clearly, in case Φ, ϕ, φ, δ are null, we find again the standard RCPSP Problem. It is possible to obtain a polyhedral formulation of the RCPSMRP and to adapt the *Connectivity* Theorem.

5.2. The *Insertion Money Flow* Problem.

As for the RCPSTDP Problem, our main algorithmic tool consists in an *Insertion* mechanism, which will allow us, at any time, to consider some Timed (r*, ϕ*, φ*)-Flow (*F*, *F*-cash, *T*) defined on some subset $W \subset V$ of the task set *V*, and try to *insert* in an optimal way some task $v_0 \in V - W$ into (*F*, *F*-cash, *T*). In order to formalize this process, we do as in Section 3: we deal with standard renewable resources $k \in K$ while considering the *Insertion Flow* model of Section 3, together with null conditional delay *D-Lag* values, and we introduce the *Insertion Money Flow Problem* which is specifically related to the management of the *Money* resource.
An *Insertion Money Flow* instance ((*X*, *E*), z_0, *A*, *B*, *In*, *Out*, Π, Π*, τ, ρ, γ, π, d) is then defined by an almost-bipartite graph *N* = (*X*, *E*), by z_0, *A*, *B*, *In*, *Out* , Π, Π* as in Section III, by a **Q**-valued *payment delay* function $\tau \geq 0$ with domain *A*, and by four non negative coefficients *d*, which means the duration of task z_0, π and ρ, γ, such that: (P8)

- $\Sigma_{x \in A} Out_x \geq \rho$; ρ represents the cash amount which is required in order to start task z_0;
- $\Sigma_{x \in A} Out_x + \gamma - \rho = \Sigma_{y \in B} In_y$; γ represents the cash amount which is produced at the end of task z_0, and available after a delay equal to π;

Then $G = (G_{x,y} \geq 0, x \in A \cup \{z_0\}, y \in B \cup \{z_0\}) \geq 0$, is an *Insertion Money Flow vector* iff:

- for any *x* in *A*, $Out_x = \Sigma_{y \in B \cup \{x0\}} G_{x,y}$; for any *y* in *B*, $In_y = \Sigma_{x \in A \cup \{x0\}} G_{x,y}$;
- $\gamma = \Sigma_{y \in B} G_{z0,y}$; $\rho = \Sigma_{x \in A} G_{x,z0}$;

For such an *Insertion Money Flow* vector *G*, we extend Π and τ by setting:

- $\Pi_{z0} = Sup(Sup_{x \in A, \text{ such that } (x, z0) \in E} (\Pi_x + d), Sup_{x \in A, \text{ such that } Gx, z0 \neq 0} (\Pi_x + \tau(x) + d))$; (in case no flow $G_{x,z0}$ exists, we set $\Pi_{z0} = d$);
- $\tau(z_0) = \pi$;

and we define the *Makespan* value *F-Makespan*(*G*) by setting:

- *F-Makespan*(*G*) = Sup ($Sup_{x \in A \cup \{z0\}, y \in B \text{ such that } (x, y) \in E} (\Pi_x + \Pi^*_y)$, $Sup_{x \in A \cup \{z0\}, y \in B \text{ such that } Gx,y \neq 0} (\Pi_x + \Pi^*_y + \tau(x))$).

Then solving the related *Insertion Money Flow Problem* consists in computing a related *Insertion Money Flow* vector *G* in such a way that *F-Makespan*(*G*) be the smallest possible.

The general algorithmic *Insertion-Money-Flow* Procedure works according to the same scheme as *Insertion-Flow* of Section 3. Still, it behaves in a simpler way. As a matter of fact, we may notice that, once an attachment value *u* has been chosen in the set $\Lambda_{E, z0} = \{u \in \mathbf{Q}$, such that:

- there does not exist $x \in A$, such that $\Pi_x > u$ and $(x, z_0) \in E$;
- there exists $x(u) \in A$, such that $\Pi_{x(u)} + \tau(x(u)) = u$;
- $\Sigma_{x \in A, \text{ such that } \Pi x + \tau(x) \leq u} Out_x \geq \rho\}$

then the related *Insertion Flow G* may be obtained through a greedy process:

- Perform the *attachment process*: compute values $G_{x,z0}$, $x \in A$ in such a way that: (P9)
  - if x and x' are such that: $G_{x',z0} \neq 0$, $\Pi_{x'} + \tau(x') < \Pi_x + \tau(x) \leq u$, then: $G_{x,z0} = Out_x$;
  - for any x such that: $\Pi_x + \tau(x) > u$ then: $G_{x,z0} = 0$;
- Apply a greedy *Match-Flow* procedure, which implements the following lemma:

Match Flow Lemma*: If the No Cross Property is satisfied, that means if there does not exist $x, x' \in A \cup \{z_0\}$, $y, y' \in B$, such that: $G_{x,y} \neq 0$; $G_{x',y'} \neq 0$; $\Pi_x' + \tau(x') > \Pi_x + \tau(x)$; $\Pi^*_{y'} > \Pi^* y$, then G is optimal (under the constraint that the attachment value u has been imposed)*

We may state the following result:

Theorem 6: *The Insertion Money Flow Problem is Time Polynomial.*

Proof: it follows the lines of the proof of Theorem 3.

As in Section 3, every time we insert some task v_0 into some *Timed (r^*, ϕ^*, φ^*)-Flow* (F, *F-cash*, T) defined on some subset $W \subset V$, while using some cut (U, $W - U$), we apply the *Insertion Flow* and *Insertion Money Flow* procedures respectively to every renewable resource and next to the *Money* resource. In order to increase the efficiency of the process, we make (*Lex-Insertion-Money-Flow* Procedure) the flow values $F(k)_e$ be supported by the same arcs.

5.3. Generic Flow Algorithms for the RCPSFRP Problem.

The generic machinery of Section 3 can be used again here, and gives rise to a Greedy Insertion Algorithm **RCPSMRP-Greedy-Flow**, which may be randomized, and to a Local Search **RCPSMRP-LS-Flow** Algorithm, Of course, both may be combined into a GRASP algorithmic scheme. Still, there is a specific point which has to be discussed here. As told in 5.1, testing the feasibility of some RCPSMRP instance is NP-Complete. That means that, when we try the insertion of some task v_0 into some into some Timed (r*, ϕ*, φ*)-Flow (F, *F-cash*, T) defined on some subset $W \subset V$, there is a risk of failure, since, for any cut U, there may be not enough cash coming from U in order to fill the demands from both v_0 and W - U.

As a consequence, we see that the way we pick up tasks v of V inside the greedy **RCPSMRP-Greedy-Flow** scheme becomes a key issue. We call it the "*Linear Ordering*" component of the RCPSMRP Problem. In order to deal with it, we associate with any RCPSPMRP instance $\boldsymbol{I}$ = (V, K, R, r, d, $<<$, Φ, ϕ, φ, δ), the following auxiliary problem PLIN(***I***): {Compute a linear ordering σ of V such that: for any $v \in V$, $\Phi + \Sigma_{w \text{ such that } w \sigma v} (\varphi(w) - \phi(w)) \geq \phi(v)$}.
Then we notice that:

Theorem 7: *Let T be some feasible solution of the RCPSPMRP Problem instance $\boldsymbol{I}$ = (V, K, R, r, d, $<<$, Φ, ϕ, φ, δ). Then, any linear ordering σ which is compatible with T, d and δ (that means such that if $T(v) + d(v) + \delta(v) \leq T(w)$ then $v \sigma w$) is a solution of the PLIN(**I**) Problem). Conversely, if we apply the **RCPSMRP-Greedy-Flow** algorithm to **I** while picking up the tasks of V according to such a feasible solution σ of PLIN(**I**), then we are sure to get a feasible solution of **I**.*

So, we derive from this result the following way to handle the *Linear Ordering* component of the RCPSMRP Problem while designing the **RCPSMRP-Greedy-Flow** algorithmic scheme:

- Compute a set Λ of N distinct solutions of PLIN(***I***), N being some replication parameter which is small enough in order to make possible the computation of Λ; do in such a way to get solutions which are not too close to each other;
- For any σ in Λ, apply the **RCPSMRP-Greedy-Flow** algorithm while picking up the tasks v of V according to the linear ordering σ;
- Keep with the best solution $T = T(\sigma)$, $\sigma \in \Lambda$, which was produced during the *For* loop.

We adapt the local search **RCPSMRP-LS-Flow** algorithmic scheme by the same way:

- Starting from a current feasible solution T of the RCPSMRP instance ***I***, we first compute a linear ordering σ which is compatible with T, d and δ;
- Next, every time we perform the removal of some subset W of V, we compute a set Λ of M distinct linear orderings σ^* of V, which coincide with σ on $V - W$ and which are solution of PLIN(***I***), M being some parameter value;
  - We denote by Λ_W the set of the restrictions of σ, σ in Λ, to W;
  - For every linear ordering τ in Λ_W, we try the reinsertion of W while picking up the tasks of W according to τ (Failure may occur).

5.4. Numerical Experiments.

We provide here results related the behaviour of the **RCPSMRP-Greedy-Flow** Procedure. We used 30 and 60 task instances ***I*** from PSPLIB, together with some solution T(***I***) obtained from the tests of Section 4. For every such an instance ***I***, we introduced *Money* and generated values $\phi(v)$, $\varphi(v)$, $\delta(v)$, $v \in V$, and Φ, in such a way that, for every instance ***I***, T(***I***) remained a feasible solution with unchanged makespan MK(***I***). We denoted by *Gap-FR* the gap (in %) between the value obtained by the **RCPSMRP-Greedy-Flow** Procedure and the almost-optimal value MK(***I***), and got results according to the following table:

Table 6: RCPSFRP-Greedy-Flow Behaviour, Mean Results.

N-Ac	*N-res*	*N-re*	*Time*(*s*)	*Gap-FR*
30	4	10	0.05	9.7
30	4	100	0.49	4.8
30	4	1000	5.3	3.7
60	4	10	0.61	13.8
60	4	100	6 .7	5.9
60	4	1000	73.7	4.8

Comment: *Money* resource makes the problem more difficult: the *Linear Ordering* component is really a critical issue, and randomization becomes really significant.

6. Scheduling through Borrow/invest Strategies

In this extension of the RCPSMRP, we suppose that, at any time during the project, we are able to borrow money at a constant rate ρ_B and to invest money at a constant rate ρ_I in order to speed up our project. We call *Investment plan* any triple (t, Δ, M) whose meaning is: an

amount M of money is invested at time t, which gives back $M.(1+ \rho_I).\Delta$ money units at time $t + \Delta$. We denote by *Invest* the set of all *Investment* plans. By the same way we call *Loan plan* any triple (t, Δ, M) whose meaning is: an amount M of money is borrowed at time t, and $M.(1+ \rho_B).\Delta$ money units are given back at time $t + \Delta$. We denote by *Loan* the set of all *Loan* plans.

Let us consider some RCPSMRP instance $\boldsymbol{I} = (V, K, R, r, d, <<, \Phi, \phi, \varphi, \delta)$, and let us denote by $\boldsymbol{I}(\Psi)$ the instance we get by replacing Φ by $\Phi + \Psi$. For a given value Ψ, let $S = (F, F\text{-}Cash, T)$ be some solution of $\boldsymbol{I}(\Psi)$, which induces a T_{End} value. At any time t in $[0, T_{End}]$, we are provided with an amount $Cash_S(t)$ of available money. $Cash_S(t)$ is the quantity $M(T, t)$ which was involved in the definition of the RCPSMRP. While the function $t \rightarrow Cash_S(t)$ is a positive piecewise constant function, the function $t \rightarrow Cash_S(t) - \Psi$ may take negative values, which forbids $(F, F\text{-}Cash, T)$ from being viewed as a feasible solution of $\boldsymbol{I} = \boldsymbol{I}(0)$.

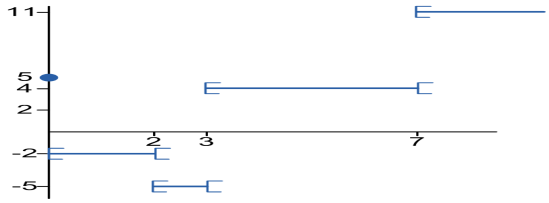
Still, we may think into using convenient sets $Inv \subset Invest$ of Investment plans and $Lo \subset Loan$ of loan plans, all starting and ending in $[0, T_{End}]$, in order to *balance* the $t \rightarrow Cash_S(t) - \Psi$ function, that means to make in such a way that, at any time t in $[0, T_{End}]$, the induced *Available Cash* $Cash_S(t) - \Psi + Inv_0(t) + Lo_1(t) - Inv_1(t) - Lo_0(t) \geq 0$, where:

- $Inv_0(t)$ = Money produced by the plans in *Inv* which end at time no larger than t;
- $Inv_1(t)$ = Money required by the plans in *Inv* which start at time no larger than t;
- $Lo_0(t)$ = Money given back by the plans in *Lo* which end at time no larger than t;
- $Lo_1(t)$ = Money obtained from the plans in *Inv* which start at time no larger than t.

Thus, solving the *Second Resource Constraint Project Scheduling Problem with Money Resource* defined by $\boldsymbol{I} = (V, K, R, r, d, <<, \Phi, \phi, \varphi, \delta)$ and by the interest rates ρ_B and ρ_I, means computing: Ψ, $(F, F\text{-}Cash, T)$, *Inv* and *Lo* in such a way that the $t \rightarrow Cash_S(t) - \Psi$ *Available Cash* function be *balanced*, and T_{End} be the smallest possible.

Example: we consider the example of the previous section, with $\rho_B = 50\%$, $\rho_I = 20\%$, null payment delay and an initial *Cash* amount equal to 5. We get the *Available Cash* function :

Figure 3: The *Available Cash* function with Initial Cash Amount = 5;



We balance this function through: First *Loan* plan: (0, 2, 2); Second *Loan* plan: (2, 1, 7.5); Last *Investment* plan: (3, 4, 2.25); the *Val* value of the *Balancing* procedure below is equal to 0.6.

The search for *Inv* and *Lo* can be done according to the following result:

Theorem 8: *There exists Inv and Loan which balances the Available Cash function* $t \rightarrow Cash_S(t) - \Psi$ *if and only if the following balancing procedure returns a positive Success value.*

Balancing Procedure

Input: the interest rates ρ_I, ρ_B, the function t -> $Cash_S(t)$ and the number Ψ;
Output: *Success*: Boolean, *Lo*: Loan subset, *Inv*: Investment subset;
Main Loop:
t-cour <-0; *Inv* = *Lo* <- Nil; *Free-Cash* is the function t -> $Cash_S(t) - \Psi$; *Success* <- 0;
While *t-cour* $\neq T_{End}$ do
Let δ be the largest value in [*t-cour*, T_{End}] such that *Free-Cash* is constant on [*t-cour*, *t-cour* + δ[and α be the related value;
If $\alpha > 0$ then
Insert the investment I = (*t-cour*, δ, α) in *Inv* and update *Free-Cash* accordingly;
Else if $\alpha < 0$ then
Insert the loan *L* = (*t-cour*, δ, - α) in *Lo* and update *Free-Cash* accordingly;
Val <- *Free-Cash*(T_{End}); If *Val* ≥ 0 then *Success* <- 1;

So, we handle the 2 th RCPSMRP Problem through the following dichotomic scheme:

Second-Money-Flow Algorithm

Input: $\boldsymbol{I}$ = (*V*, *K*, *R*, *r*, *d*, <<, Φ, ϕ, φ, δ), together with the interest rates rates ρ_I, ρ_B;
Output: a value ψ, a solution (*F*, *F-Cash*, *T*) of $\boldsymbol{I}(\psi)$, and *Borrow/Invest* plans *Lo*(Ψ), *Inv*(Ψ);
Main Loop:
ψ-*min* <- 0; Compute a solution (F_{min}, $F\text{-}Cash_{min}$, T_{min}) of $\boldsymbol{I}(\psi\text{-}min)$;
ψ-*max* <- $\Sigma_{v \in V} \phi(v) - \Phi$; Compute a solution ($F_{max}$, $F\text{-}Cash_{max}$, T_{max}) of $\boldsymbol{I}(\psi\text{-}max)$;
If the *Balancing* Problem related to Ψ-*max* admits a solution *Inv*, *Lo* then
Second-Financial-Flow <- (F_{max}, $F\text{-}Cash_{max}$, T_{max}), ψ-*max*, *Lo*, *Inv*
Else
Not Stop; *Current-Solution* <- (F_{min}, $F\text{-}Cash_{min}$, T_{min}), 0, Nil, Nil;
While Not stop do
Ψ-*medium* <- (Ψ-*max* + Ψ-*min*)/2;
Compute a solution (F_{medium} $F\text{-}Cash_{medium}$, T_{medium}) of $\boldsymbol{I}(\psi\text{-}medium)$;
If the *Balancing* Problem related to Ψ-*medium* admits a solution *Inv*, *Lo* then
Ψ-*min* <- Ψ-*medium*; *Current-Solution* <- (F_{medium} $F\text{-}Cash_{medium}$, T_{medium}), Ψ-*medium*, *Inv*, *Lo* Else Ψ-*max* <- Ψ-*medium*;
If Ψ-*max* - Ψ-*min* is small enough then Stop;
Ψ <- ψ-*min*.

We performed numerical tests with focus on the way the *Makespan* value depend on the ρ_B, ρ_I values. We selected some 30 task instances of PSPLIB, and randomly generated parameters Φ, ϕ, φ, δ in such a way that those instances still admitted a feasible solution. For every instance, we computed the gap *GAP-F* = (*V-1* – *V-Standard*)/*V-Standard* between the standard RCPSP optimal value *V-Standard* and the value *V-1* obtained through **RCPSMRP-Greedy-Flow** (with 10 replications), and the gap *GAP-Rate* = (*V-1* – *V-Rate*)/*V-Rate* between the value *V-1* and the value *V-Rate* obtained with the **Second-Money-Flow** Procedure with related rates ρ_B, ρ_I .

We got results which are summarized in the following tables:

Table 7: Impact of *Borrow/Invest* strategies: Procedure *Second-Financial-Flow*

ID	ρ_I	ρ_B	*GAP-F (%)*	*Gap-Rate (%)*
1	0.1	0.2	73	67
2	0.3	0.5	24	15
3	0.5	0.7	5.2	35
4	0.3	0.3	98	28
5	0.5	0.5	19	19
6	0.1	0.2	125	65
7	0.3	0.5	34	32
8	0.5	0.7	57	27

Comment: In case, *Money* constraint significantly slows down the project, ad hoc *Borrow/Invest* strategies may ease this *Money c*onstraint.

6. Conclusion

We have been introducing here flow models in order to deal with different extensions of RCPSP. The main interest of those flow models relies on their very generic features, which allow fast software implementation. Several questions would be worth receiving more interest, from both theoretical and practical point of view: they are related to the way *Cuts* are generated and about what we called the *Linear Ordering* component of our algorithms.

References

1. R. Kolisch, R. Padman. “An integrated survey of deterministic project scheduling”, Omega, 29, pp. 249-272 (2001).
2. P. Brucker, A. Drexl, R. Mohring, K. Neumann, E. Pesch. “Resource-constrained project scheduling: notation, classification, models and methods”, *EJOR* 112, pp. 3-41 (1999)
3. P. Baptiste, P. Laborie, C. Lepape, WP.M. Nuijten. “Constraint-based scheduling and planning”, In F. Rossi, P. van Beek & T. Walsh (Eds.), *Handbook of Constraint Programming*, pp. 761-799. Amsterdam: Elsevier (2008).
4. N.W. Sauer, M.G. Stone, “Rational preemptive scheduling”, *Order* 4, pp. 195-206, (1987)
5. J. Blazewiecz, K.H. Ecker, G. Schmidt, J. Weglarz. *Scheduling in computer and manufacturing systems*, 2 th edn, Springer-Verlag, Berlin (1993)
6. W. Herroelen. “Project scheduling-theory and practice”, Productions and Operations Management., 14, 4, pp. 413-432 (2005)
7. S.S. Liu, C.J. Wang, “Resource-constrained construction project scheduling model for profit maximization considering cash flow, Automation in Construction 17, 8, pp. 966-974 (2008)
8. J. Damay, A. Quilliot, E. Sanlaville. « Linear programming based algorithms for preemptive and non preemptive RCPSP”, *EJOR*, 182, 3, pp. 1012-1022 (2007)
9. A. Moukrim, A. Quilliot. “Preemptive scheduling on parallel machines”, *O.R Let.*, 33, pp. 143-51 (2005)
10. B. De Reyck, W. Herroelen. “A Branch and bound procedure for the resource-constrained project scheduling problem with generalized precedence relations”, *EJOR* 111, 1, pp. 152-174 (1998)
11. R.H. Mohring, F.J. Radermacher. “Scheduling problems with resource duration interactions”, *Methods Operat. Research* 48, pp. 423-452 (1984)

12. A. Kimms. *Mathematical programming and financial objectives for scheduling projects*, Operations Research and Management Sciences, Kluwer Academic Publisher (2001)
13. H. Chtourou, M. Haouari. "A two-stage-priority rule based algorithm for robust resource-constrained project scheduling", *Computers and Industrial Engineering*, 55, pp.183-194 (2008)
14. M. Haouari, A. Gharbi. "An improved max-flow based lower bound for minimizing maximum lateness on identical parallele machines", *OR Letters* 31, pp. 49-52 (2003)
15. B. Abbasi, S. Shadrokh, J. Arkat. "Bi-objective resource-constrained project scheduling with robustness and makespan criteria", *App. Math. and Computation* 180, pp. 146-152 (2006)
16. J. Carlier, E. Neron. "Computing redundant resources for the resource constrained project scheduling problem", *EJOR* 176, pp. 1452-1463 (2007)
17. S. Hartmann, D. Briskorn. "A survey of variants and extensions of the RCPSP". *EJOR*. 207, pp. 1-14, (2010)
18. R. Kolisch, S. Hartmann. Heuristic algorithms for solving the resource constrained project scheduling problem: classification and computational analysis. In J. Weglarz, editor, Project scheduling: Recent models, algorithms and applications, pp. 147-178. Kluwer academic, Amsterdam, the Netherlands (1999)
19. E. Demeulemeester, W. Herroelen. "New benchmark results for the RCPSP", *Management Science*, 43, pp. 1485-1492 (1997)
20. R. Kolisch, A. Sprecher, A. Drexel. Characterization and generation of a general class of RCPSP, *Management Science*, 41, (10), pp.1693-703 (1995)
21. P. Brucker, S. Knust, A. Schoo, O. Thiele. "A branch and bound algorithm for the resource constrained project scheduling problem", *EJOR* 107, pp. 272-288 (1998)
22. J. Josefowska, M. Mika, R. Rozycki, G. Waligora, J. Weglarz. "An almost optimal heuristic for preemptive cmax scheduling of dependant tasks on parallel identical machines". Annals of Operations Research, 129, pp. 205-216, 2004.
23. A. Mingozzi, V. Maniezzo, S. Ricciardelli, L. Bianco. « An exact algorithm for RCPSP based on a new mathematical formulation", *Management Science* 44, pp. 714-729 (1998)
24. J.H. Patterson. "A comparizon of exact approaches for solving the multiple constrained resource project scheduling problem", *Management Sciences* 30 (7), pp. 854-867 (1984)
25. P. Baptiste, S. Demassey. "Tight LP-bounds for RCPSP", *OR Spect.* 26, 2, pp. 251-262 (2004)
26. P. Brucker, S. Knust. "A linear programming and constraint propagation based lower bound for the RCPSP", *EJOR* 127, pp. 355-362 (2000)
27. C. Artigues, F. Roubellat. "A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple nodes", *EJOR* 127-2, pp. 297-316 (2000)
28. C. Artigues, P. Michelon, S. Reusser. "Insertion techniques for static and dynamic resource-constrained project scheduling.", *EJOR* 149, pp. 249-267 (2003)
29. K. Bouleimen, H. Lecocq. "A new efficient simulated annealing algorithm for the RCPSP and its multiple mode version", *EJOR* 149, pp. 268-281 (2003)
30. R. Kolisch, S. Hartmann. "Experimental investigation of heuristics for the resource constrained project scheduling: an update"; *EJOR* 174, pp. 23-37, (2006)
31. R.V. Ahuja, T.L. Magnanti, J.B. Orlin. "Network Flows: Theory, Algorithms, and Applications", Prentice Hall (1993)
32. U. Dorndorf, E. Pesch, T. Phan-Huy: "Constraint propagation techniques for the disjunctive scheduling problem", *Artificial Intelligence* 122, pp 189-240, (2000)

MISTA 2013

Container Stowage and Short Sea Ship Routing Problem Optimization

Ana Moura, Jorge Oliveira

Abstract: Short sea shipping has several advantages over other means of transportation, recognized by EU members. The maritime transportation could be dealt like a combination of two well-known problems: the container stowage problem and routing planning problem. The integration of these two well-known problems results in a new problem CSSRP (Container stowage and ship routing problem) that is also an hard combinatorial optimization problem. In this work we prove that regardless of the complexity of this problem a mixed integer mathematical programming model (MIP) could be used to generate optimal solutions in a reduced computational time. The main goal is to solve the container stowage problem and also to plan the vessels routes in an integrated way. In dealing with this integrated problem it is expected that the management of small fleets of containerships of short sea distribution will be optimized and the transportation costs are reduced. For testing the mathematical model some problems based in real data were used and the optimality was always achieved in a very short computational time.

1 Introduction

There are several transportation systems that can be used to transport containers from one destination to another. Transport over sea is among the various forms of transportation and the one with a greatest growth rate over the last decades. Presently, short sea shipping is responsible for a significant part of all freight moved within the European Union borders. In the last decades containerization has revolutionized cargo shipping. According to Drewry Shipping Consultants, today over 70% of the value of the world international seaborne trade is being moved in containers. The world container fleet expanded at an average annual growth rate of 9%. A significant part of all freight moved within the European Union travels by sea. EU statistics state that 3800 million tons were transported by ship in 2006 and it is hoped that, in 2018, if the economic crisis overcome, the 5300 million tons mark will be reached. Short

Ana Moura
Department of Economics, Management and Industrial Engineering, University of Aveiro
INESC Coimbra - Institute for Systems Engineering and Computers at Coimbra
E-mail: ana.moura@ua.pt

Jorge Oliveira
Department of Economics, Management and Industrial Engineering, University of Aveiro
E-mail: jorge.antonio@ua.pt

sea shipping has several advantages but also has some downsides: the bureaucracy attached to customs and ports; port services costs and efficiency; travel duration; inflexibility of routes; and dependency on environmental factors. In order to reduce some of these downsides, this work intends to optimize and minimize the transportation costs reducing the travel duration and the port servicing cost and efficiency. With that purpose, a mixed integer linear programing (MIP) model that minimizes the maritime transport distribution costs was developed.

The motivation and the idea for this work arise from the parallelism between this type of distribution and the distribution problems over land. In this latter type of transportation a well-known problem, the Vehicle Routing and Loading Problem (VRLP) that is an integration of the Vehicle Routing Problem (VRP) and Container Loading Problem (CLP), was first presented by Moura and Oliveira [1]. The routes and the packing of boxes (or pallets) in the trucks are computed considering the Last-In-First-Out (LIFO) strategy and all the constraints related to this two different problems. The study of VRLP only started in 2006 and since then further work has been done. Gendreau et al. [2] and Moura and Oliveira [1] were the first authors to integrate the VRP and CLP and to publish benchmark instances. Moura and Oliveira [1] considered a mathematical formulation to the VRLP with Time Windows and tri-dimensional packing and in this work they presented two heuristic approaches, a hierarchical and a sequential approach. Bearing this in mind, we applied the same concept to short sea distance distribution problems. The routes are planed depending on: distances, demands and the delivery deadlines of each port. Besides, the mathematical model considers the interconnection between the routes optimization and the containers loading on vessels, in order to prevent overstowing[1]. According to these, two different kinds of optimization problems must be dealt with, in an integrated manner: the Vessel Routing Problem with Deadlines and the Container Stowage Problem. The integration of these two problems was named by Moura et al. [3] as the Container Stowage and Ship Routing Problem (CSSRP).

This paper is structured as follows: section 2 consists of a detailed description of the CSSRP; section 3 is a review of the relevant literature; in section 4 a mixed integer programming model of CSSRP is presented; section 5 is dedicated to the discussion of results, considering some generated problem instances, and the analysis of the model performance; to conclude in section 6, some global remarks and suggestions for future work are presented.

2 Container Stowage and Ship Routing Problem description

The containerships are vessels specifically constructed to transport containers. The cargo space of the vessel is divided in bays positioned on deck and below the deck of the ship and separated by a hatch cover [4]. Each bay (longitudinal sections) consists of a set of stacks (transversal sections) and tiers (vertical sections of a stack) that are one container wide (Figure 1). The space related to one stack and tier is usually referred by a slot, which is a single space where containers could be loaded. The vessels have capacity constraints, related to weight, slot dimensions and number of slots. Although there are different types of containers with different dimensions and characteristics, in this work only the containers types considered in our problem are described. Each container has a weight, a destination port and standard dimensions. Those standard dimensions are measured in foot equivalent units and could be TEU (Twenty-foot Equivalent Unit) or FEU (Forty-foot Equivalent Unit). In each slot one 40´ or two 20´ can be loaded. Another type of containers are those that need electric power (refrigerated containers). These particular containers must be loaded in slots that have a power plug. The maximum weight of each container is 24 tons for 20´and 40 tons for 40´.

[1] An overstow occurs when there are containers that must be moved because they block the access to other containers that have to be unloaded.

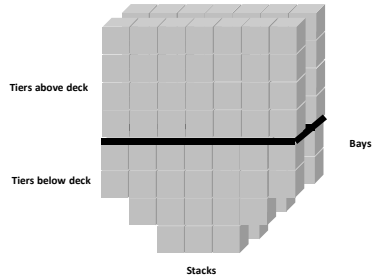


Figure 1 - Containerships cargo positioning

In CSSRP we have a set of ports each one with a demand (set of containers) that must be delivered by a vessel within a given period of time (deadlines). Two major decisions must be considered: which ports should be visited by each vessel and the visit sequence, and how to stow the containers on board when considering all the placement requirements and LIFO strategy. The goals are to determine the best visiting sequence of ports and to prevent overstowing of containers, while minimizing the total distribution cost.
As explained in [3] the short sea shipping main costs are related to the containers handling in the ports. So the position of the containers on the vessels is crucial to reduce time spending in a port to unload them. It is well known that the vessel's stowage plan influences port handling. So, we consider that a movement is performed when: containers are unloaded because they reached their destination and containers have to be moved (shift) because they block the access to others that have to be unloaded (overstow). There is another reason to move a container, it is when the containers must be re-positioned to improve the overall stowage (vessel stability) or/and make the next port handling easier, which are known as re-handles. These last movements are not considered in this approach. Shifting is a time consuming activity, hence the arrangement of containers on board is crucial to achieve operational efficiency and reduce the number of overstows. It is easy to understand that when a shift (which always implies two movements) is performed the container is always loaded in a different slot and the stowage plan is normally rearranged. To reduce the number of shifts, the LIFO strategy is considered when the routes are computed. Because the number of overstows depends of the route visiting sequence, which is part of the decision making process. Beyond the costs implicit on a movement, there are other costs that have to be considered: costs related to ports, taxes and utilization costs; related with the vessels, for example the cost of travel operation that depends on the cost of the fuel/mile and the cost of vessel utilization which depends on crew member per day. So, in order to reduce the global short sea distribution cost we compute an optimal route that minimizes a two component function of the distribution costs and overstowing costs.
The CSSRP has several constraints that have to be considered. The constraints are divided in three groups: related to routes/ports (C1), related to vessels/loading (C2) and those that are related to both (C3).

(C1) The Routing constraints:

Deadline constraint: deadline limits can be modelled as linear vehicle routing constraint, where the arrival time of the vessel to a port must be less than or equal to the smallest container deadline of the port demand.

(C2) The Loading and Placement constraints:

1. Vessel capacity constraints: maximal weight and number of containers. Weight limits and the number of slots, can be modelled as a linear knapsack constraints, where the number of containers loaded and sum of its weights must be smaller or equal to the vessel capacity;

2. Positioning constraints, related to container placement stability in vessels. Can be modelled as a three-dimensional container loading problem constraints, where containers could be loaded directly on the vessel floor (or on the hatch cover) or above another container(s).

(C3) The Routing and Loading constraints:
LIFO strategy, which is directly related to the overstow constraints.

The challenge develops a MIP model for the CSSRP that can achieve optimal solutions in a reduced computational time and can be applied to real problems, is tackled in this study, being the main contribution of the present work.

3 Literature Review

The CSSRP has received little attention in literature. All the approaches we were able to find have only considered the ships' container stowage problem (CSP), sometimes considering predefined port visiting sequence. Nevertheless, as far as we know the first published work that deals with the Container Stowage Problem (CSP) and the well-known Vehicle Routing Problem (VRP) with deadlines in an integrated way is from Martins et al. [7]. Later on, Moura et al. [3] proposes a MIP model that solves the CSSRP but with some assumptions and simplifications in order to simplify the model. These simplifications are related to the containers. The distinction between containers (TEUs or FEUs and refrigerated) are not considered. In this model a demand to a given port is addressed like one big container with a defined dimension and a weight, that corresponds to the summation of dimensions and weights of all container's demand. So, the placement matrix is simplified and reduced. In this present work and with the aim to improve and adapt the model of [3] to real problems, these simplifications are not taken in consideration.
Regarding the ship routing and scheduling, Christiansen and Nygreen in [5] present an optimization-based solution approach for a real vessel's planning problem which the authors characterized as the Inventory Pickup and Delivery Problem with Time Windows (IPDPTW). The mathematical programming model was solved using the Dantzig–Wolfe decomposition, imposing the original formulation into a subproblem for each port and each ship. The linear programming relaxation of the master problem is solved by column generation, where the columns represent the vessel's routes or port's visit sequences. In order to make the integer solution optimal, the iterative solution process is embedded in a Branch-and-Bound search. Later on, Agarwal and Ergun [6] presented an integrated MIP model to solve the vessel´s scheduling and cargo-routing problems simultaneously. In this work a greedy heuristic, a column generation based algorithm and a two phase algorithm were developed. This approach is able to generate good schedules for vessels. Later on Martins et al. [7] proposes a genetic algorithm approach to improve the flexibility of short sea shipping and to increase its competitiveness with other means of freight transport. A logistic model was developed to manage a fleet of vessels which transport cargo to and from several ports, bearing in mind the cargo loading and delivery deadlines. The results show that the overall efficiency of short sea shipping can be improved.
In the last three decades unlike CSSRP, several works on CSP, also known as the Master Bay Problem (MBP), were published. Considering the description of the CSSRP in section 2, is easily understandable that the CSP is a NP-hard problem [8]. This problem concerns the task of determining the arrangement of containers in a containership. It can be categorized as an assignment problem where a set of containers with different characteristics and with a given port destination, must be assigned to slots in a vessel aiming to minimize the transportation cost. Since the late 80's there were several works published about the CSP. To the best of our knowledge Aslidis [9] was the first author to solve the stack overstowage problem using a dynamic programming algorithm. The goal was the arrangement policy and this approach was widely adopted in later works. Avriel and Penn [10] proposed the whole columns heuristic procedure to find the optimal solution for a stowage problem in a single rectangular bay with only accessibility constraints. This heuristic requires solving a MIP model after some pre-processing of the data. This method was proven to be limited because of the large number of binary variables and constraints needed to the formulation. Later on, Avriel et al. [11]

developed the suspensory heuristic procedure that achieves very satisfactory results in a short computation time. Nonetheless, the method proved to be very inflexible as far as the implementation of constraints is concerned. Binary linear programming formulations for the CSP with stability constraints, weight constraints, accessibility constraints, etc., can be found in [12], [13] and [14]. In all of these works, the authors concluded that it is impossible to obtain optimal solutions through MIP for these problems with additional constraints. A very recent work that solves the vessel stowing planning, from Delgado et al. [15] contradicts this statement. They decompose the problem and present a constraint programing and integer programming model for stowing a set of containers in a single bay section and solve real life problems to optimality in a reasonable computational time.
Several searching methods such as Genetic Algorithms (GA) [16], [7] and tabu search [17] have also been applied to CSP. The advantage of using heuristic and meta-heuristic approaches to deal with this kind of problems has been proved with these works. Wilson and Roach in [17] and later on in [4], presented a two-phase method. In the first stage containers are grouped by their destination using a Branch-and-Bound search algorithm, aiming to reduce overstows and hatch movements at the next port-of-destination. After that, in the second stage, a tabu search algorithm is applied to the generalized solution, trying to move the containers and assigning them to a specific slot, in order to reduce re-handles, bearing in mind the stability constraints.
The first time that the CSP was compared and characterized like another well-known problem – the Bin Packing Problem - was by Wei-Ying et al. [18]. In this work the CSP is regarded as a bidimensional packing problem, where the bays of the vessels are regarded as bins, the number of slots in each bay is regarded as the capacity of the bins, and containers with different characteristics are treated as items to be packed. A two stage approach was developed: in the first stage two objective functions were considered, one to minimize the number of bays packed by containers and the other to minimize the number of overstows. Then the containers assigned to each bay in the first stage are allocated to special slots in the second stage, applying a tabu search algorithm. Constraints like weight, stability and overstows are considered.

4 The MIP Model

In this section a mixed integer linear programming model, aimed to guarantee the generation of optimal solutions to the CSSRP is presented. A mathematical model was developed to manage a fleet of vessels which must transport the container's demands to several ports, bearing in mind the cargo loading and delivery deadlines. The data and variables of the model are described in the following sections.

4.1 CSSRP data

Each port has a different geographical location and is represented by a set of nodes in a direct graph $G(P, A)$, where $P = \{1,..., p\}$ represents the set of ports and $A = \{(i, j) : i, j \in P, i \neq j\}$ the set of edges in G. The length of each arc d_{ij} corresponds to the distance between port i and j in miles. u_{ik} is the visiting cost (in euros) of port i by vessel k. This cost depends on the tax and utilization costs of each port. Let us assume that the initial port for vessel departure is port one and the visiting cost in this port is also considered. We consider a vessel's fleet, represented by set $V = \{1,..., v\}$. For the sake of simplicity, it is assumed that the navigation speed (vel_k) is constant throughout all the travel distance. c_k is the vessel's traveling operation cost (per unit distance, in euros) which depends

on the cost of fuel per mile; and uc_k a vessel utilization cost (per day, in euros), which depends on the crew member number. Q_k is the weight capacity of the vessel k and αMax_k is the capacity in terms of number of containers of vessel k (TEU and FEU). t_{ik} is the estimation of service time of vessel k in port i.

Another set of data related to containers must be presented. First we consider that all the containers have standard dimensions (TEU and FEU). Second, we consider that there are two types of containers: ones that we call normal containers and the refrigerated containers. The initial port has a set of containers that must be delivered to other ports-of-destination ($P = \{2,..., p\}$). Each port i has a given demand that is $\alpha_i = \alpha t_i + \alpha f_i + \alpha rt_i + \alpha rf_i$, were $\alpha t_i, \alpha f_i, \alpha rt_i, \alpha rf_i$, are the number of normal containers of 20' and 40', refrigerated containers of 20' and 40', respectively. $\alpha Total_i = \sum_{i \neq 1}^{p} \alpha_i$ is the total demand. Also, for each port the demands' weight given by qt_i, qf_i, qrt_i, qrf_i is the total weight of normal containers of 20' and 40', refrigerated containers of 20' and 40', respectively. Moreover, the demand of each port is characterized by a deliver deadline (dl_i).

The last set of data is related to the vessels loading and slots. As mentioned in section 2 and due to the irregular configuration of the cargo space in a vessel, for the bays are defined by a matrix $posM_{kcs}$ of slots (Figure 2). The matrix has a set of rows (tier) $C = \{1,...,c\}$ and a set of columns (stacks) $S = \{1,...,s\}$. Each cell of $posM_{kcs}$ has assigned a value (0 or 1) that indicates if it is a vessels' slot or not. These slots can load one container of 40´ or up to two containers of 20´. Another matrix $Plugs_{kcs}$ is used only to indicate the slots with electrical power (slots with value equal to 1), as mentioned before these slots are the only place where the refrigerated containers can be loaded (Figure 3).

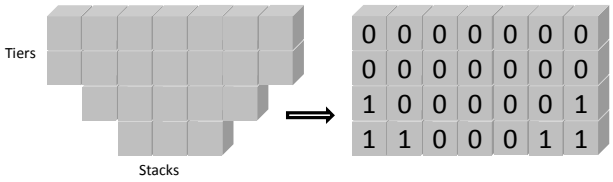


Figure 2 - Placement matrix

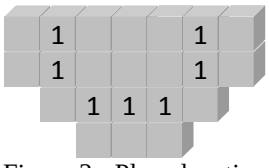


Figure 3 - Plugs location

These slots can be occupied only by one refrigerated container, because there is only one plug available. So, in this case the slot can be occupied for example, with only one refrigerated container of 40´ or with one refrigerated container of 20´ and another normal container of 20´. It should be noted that, the normal containers can also be loaded in slots with plugs, as long as

there aren't sufficient refrigerated containers to occupy them. The matrix $FinalM_{kics}$ gives us the containers' placement solution in a vessel. This decision variable is also used to integrate the routing and the stowing problems and to determine the number of overstows. As mentioned before, the cost of each movement (load or unload) is very significant in the distribution's overall cost, so m_α represents the cost incurred for loading or unloading one single container (one movement). Note that a shift involves one load and one unload. As such, in each shift (overstows occurrence) a cost of two times m_α is incurred.

4.2 The MIP model

Several decisions and auxiliary variables were considered, some of them exclusively related to the routing problem, others to the container stowage problem. With respect to the routing problem with deadlines, the first binary variables, x_{ijk} indicates if the vessel k traverses arc (i, j), or not. The other decision variable has to do with ports' deadlines, so s_{ik} is the decision variable that gives us the arrival time of vessel k to port i.

With respect to the stowage problem, another set of binary variables are used t_{kics}, f_{kics}, t_{kics}, rt_{kics}, rf_{kics}. These variables indicate if a type of container with destination port i is placed in slot (c,s) of vessel k or not. For example, t_{kics} equals one if a normal container of 20´ from ports' i demand, is loaded into a slot (c,s) in vessel k, and equal zero if not. The others are the same, but to normal containers of 40´, refrigerated containers of 20´and 40´ respectively.

Another binary decision variable is necessary to determine the number of overstows. β_{kics} indicates if the demands of ports i placed in vessel k causes overstowing or not.

The MIP model is defined as:

$$Min\left[w1*\sum_{k=1}^{v}\sum_{i=1}^{p}\sum_{j=1\wedge j\neq i}^{p}(d_{ij}c_k x_{ijk} + x_{ijk}u_{jk} + x_{ijk}\alpha_j m_\alpha) + \alpha Total_j m_\alpha + w2*\sum_{k=1}^{v} uc_k s_{1k} + w3*(2\times m_\alpha)\sum_{k=1}^{v}\sum_{j>1}^{p}\sum_{a=1\wedge a<C}^{C}\sum_{s=1}^{S}\beta_{kjas}\right] \quad (1)$$

s.t.:

C1 - Routing constraints:

$$\sum_{i=1,i\neq j}^{p} x_{ijk} \leq 1, \forall j \neq 1 \in P, \forall k \in V \quad (2)$$

$$\sum_{j=1,j\neq i}^{P} x_{ijk} \leq 1, \forall i \neq 1 \in P, \forall k \in V \quad (3)$$

$$\sum_{i=1,i\neq j}^{p} (x_{ijk} - x_{jik}) = 0, \forall k \in V, \forall j \in P \quad (4)$$

$$\sum_{j=1}^{p} x_{1jk} \le 1, \forall k \in V \quad (5)$$

$$\sum_{k=1}^{v} x_{ijk} \le 1, \forall i, j \ne i \in P \quad (6)$$

$$s_{ik} + t_{ik} + \frac{d_{ij}}{vel_k} \le s_{jk} + M(1 - x_{ijk}), \forall k \in V, \forall i, j \in P \quad (7)$$

$$s_{ik} \le dl_i, \forall k \in V, \forall i \in P \quad (8)$$

C2 - The Loading and Placement constraints:

$$\sum_{j \ne 1}^{p} (\alpha_j \times x_{ijk}) \le \alpha Max_k, \forall k \in V, \forall i \in P \quad (9)$$

$$\sum_{j \ne 1 \wedge j \ne i}^{p} ((qt_j \times \alpha t_j + qf_j \times \alpha f_j + qrt_j \times \alpha rt_j + qrf_j \times \alpha rf_j) \times x_{ijk}) \le Q_k, \forall k \in V, i \in P \quad (10)$$

$$t_{kics} + 2 \times posM_{kcs} \le 2, \forall k \in V, \forall i \ne 1 \in P, \forall c \in C, \forall s \in S \quad (11)$$

$$f_{kics} + posM_{kcs} \le 1, \forall k \in V, \forall i \ne 1 \in P, \forall c \in C, \forall s \in S \quad (12)$$

$$rt_{kics} - Plugs_{kcs} \le 0, \forall k \in V, \forall i \ne 1 \in P, \forall c \in C, \forall s \in S \quad (13)$$

$$rf_{kics} - Plugs_{kcs} \le 0, \forall k \in V, \forall i \ne 1 \in P, \forall c \in C, \forall s \in S \quad (14)$$

$$\sum_{i \ne 1}^{p} (t_{kics} + 2 \times f_{kics}) \le 2, \forall k \in V, \forall c \in C, \forall s \in S \quad (15)$$

$$\sum_{i \ne 1}^{p} (rt_{kics} + rf_{kics}) \le 1, \forall k \in V, \forall c \in C, \forall s \in S \quad (16)$$

$$\sum_{i \ne 1}^{p} (t_{kics} + rf_{kics}) \le 2, \forall k \in V, \forall c \in C, \forall s \in S \quad (17)$$

$$\sum_{i \ne 1}^{p} (rt_{kics} + f_{kics}) \le 1, \forall k \in V, \forall c \in C, \forall s \in S \quad (18)$$

$$\sum_{i \ne 1}^{p} (t_{kics} + 2 \times rf_{kics}) \le 2, \forall k \in V, \forall c \in C, \forall s \in S \quad (19)$$

$$\sum_{i \ne 1}^{p} (f_{kics} + rf_{kics}) \le 1, \forall k \in V, \forall c \in C, \forall s \in S \quad (20)$$

$$\sum_{i\neq 1}^{p}[(rt_{kics}+t_{kics}+2\times f_{kics}+2\times rf_{kics}+2\times posM_{kcs})-(rt_{kias}+t_{kias}+2\times f_{kias}+2\times rf_{kias}+2\times posM_{kas})]\geq 0,$$
$$\forall k\in V,\forall c,a<c\in C,\forall s\in S \quad (21)$$

C3 - Routing and Loading constraints:

$$\sum_{i=1}^{p}x_{ijk}\times\alpha_j=\sum_{c=1}^{C}\sum_{s=1}^{S}finalM_{kjcs},\forall k\in V,\forall j\neq 1\in P \quad (22)$$

$$x_{ijk}+\sum_{c=1}^{C}finalM_{kics}-\sum_{a<c}^{C}finalM_{kjas}\leq x_{ijk}+\sum_{c=1}^{C}finalM_{kics}-\sum_{a<c}^{C}\beta_{kjas},\forall k\in V,\forall i\in P,\forall j\neq 1\in P,\forall s\in S \quad (23)$$

The CSSRP MIP model minimizes the total route cost and the number of shifts (1). A weight is assigned to each component of the objective function components: w1, w2, w3. The concrete values of these weights will depend on the practical application under consideration and on the importance given by the decision-maker of each component. In most cases, since the total cost is usually the objective, these constants will simply be equal to 1. The first set of constraints is related to the routing problem. Constraints 2, 3 and 4 are the flow conservation constraints. Equation 2 and 3 ensures that every port is visited only by one vessel. Equation 4 ensures that if a vessel arrives to a port it also leaves the same port. Equation 5 ensures that if a vessel is used, it must begin its route in port one and equation 6 ensures that each port is visited only by one vessel. Constraint C1 (section 3), is achieved with equations 7 and 8. These ensure the feasibility of time scheduling defining the vessel setup time and the deadline constraint for serving a port, respectively. Constraint 7 guarantees that the port's service does not begin before the vessel's arrival to the port. M is a Big-M constant. The service time depends on the required time for a vessel to travel a given distance and the summation of the service time in each previously visited port (if the actual port is not the first port visited). Constraint 8 guarantees that each container deadline is not violated. For simplicity's sake, all containers with the same destination have the same deadline that will be equal to the smallest containers' deadline for that port. The following set of constraints is related to positioning and loading constraints (C2, section 3). Equation 9 and equation 10 are related to the vessels capacity. The first one states that the total demand on a route cannot exceed the vessel capacity in terms of number of containers and the second one in terms of weight. Another set of constrains are related to containers' position on vessels. As mentioned before, there are two different positioning matrixes, one related to load the normal containers and another that indicates the slots with plugs. For this reason a set of constrains, from equation 11 to equation 14, are needed in order to guarantee that the refrigerated containers are assigned only to slots with power supply and also that in each one of this slots only one refrigerated container can be placed (equations 13 and 14). On the other hand, the normal containers can be placed in any slot of the vessel and each slot can have one 40´ container or two 20´ containers. The other set of constraints (from 15 to 20) together with constraint 21 ensures the containers placement stability in vessels. Equation 15 ensures that each slot could be empty, with one 40´ container or up to two 20´ containers. Equation 16 ensures that in each slot only one refrigerated container can be placed. Equations 17 to 20 dealt with the possible positioning in a given slot between refrigerated and normal containers. These set of constraints, together with equation 21 give us the possible relative position between adjacent slots in the same tier. It guaranties that

if in a slot it is only placed one 20´ container, on top of that one 40´ container could never be placed, in order to guarantee stability. Nevertheless, it can never be placed any type of container in a slot if the slot below is empty.

The two problems (route planning and container stowage) are integrated through equation 22, that binds the container loading variables to the vehicle routing problem variables, i.e., if a vessel visits a given port then the port's demand must be placed inside that vessel. To guarantee the LIFO strategy and to minimize the number of shifts (Routing and Loading Constraint - C4, section 3), one more equation related to the overstowing constraints is needed. Equation 23 determines the number of shifts (if any), by calculating overstows' occurrence. When an overstow occurs, then a shift movement must be performed. The minimization of overstows is effectively done because the decision variable β_{kics} is minimized in the objective function.

5 Computational Results

The main goal of this approach is to solve the short sea shipping distribution problem using an exact method that can achieve a solution in a reasonable computational time. The idea is to apply this model to real-life problems. With this perspective, available data from real ships and ports has been used. The problem instances and the MIP model results are described in the following sections.

5.1 Problem instances

The problem instances were developed based on the available data of a short sea distribution problem presented in Moura et al. [3], which considers two vessels and five ports. The containership used to test the model has a capacity that could vary between 4000 and 5000 TEUs. The maximum bay size under deck is 69 TEUs (Figure 4).

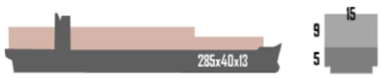


Figure 4 – Post Panamax (1988-) containership [2]

The vessels are loaded in port one and in the other ports there are only unloading or shift movements. As in [3] the problem was simplified assuming that: all port tariffs are the same; the fuel consumption and speed is always the same and independent of the cargo; some costs, such as insurance costs and others, were neglected. These simplifications only imply changes in the objective function value.
The basic problem instance is related to the short sea shipping and has 5 ports and an homogeneous demand distribution per port. This means that the number of containers and weight of each port is similar, in most of the cases equal. All the ports have deadlines (dl_i) and service time (t_{ik}), that depends on the ports' demand. These values were also computed taking into account Martins et al. [11] work. The problem instances were solved by CPLEX software, and the experiments were run on an Intel CORE i7 vPro 2,2GHz with 8Gb of memory.

[2] Adapted from http://people.hofstra.edu/geotrans/eng/ch3en/conc3en/containerships.html

5.2 MIP model results and sensibility analyses

The smaller problem has 5 ports, a set of demands for each port where the number of containers per demand is homogeneous and equal to the capacity of one vessel. The optimal solution (Figure 5) for this problem was obtained in 1,78 seconds.

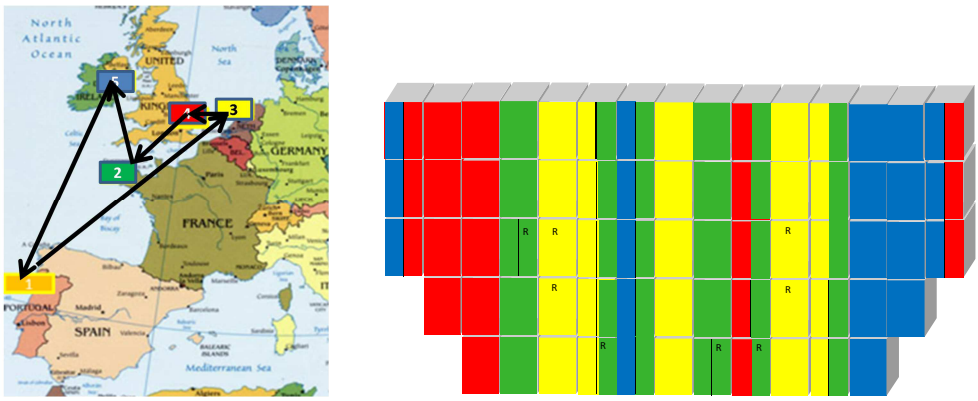


Figure 5 - Solution to five ports one vessel

As mentioned before, a sensibility analyses was performed with different and bigger instances. Our intent was to explore the effect of varying the number of ports, quantities of demands and number of vessels and deadlines. So, in order to prove the robustness and behavior of the model, three different scenarios were tested.

SA1: Increasing the number of ports and the number of vessels:
With a homogeneous demand distribution, the number of ports and vessels was increased. The results achieved are presented in table 1.

Vessels	Ports	Value Integer Optimal Solution (€)	Computational Time (seconds)	GAP (%)	Number of variables	Number of Constraints
1	5	143955,34	1,78	59,4	2131	8193
2		239637,75	94,43	89,7	4261	15150
1	10	221348,67	3,25	53,6	4461	18068
2		-	>15 minutes	-	8921	33310

Table 1 - Computational results varing number of the ports and vessels

It can be observed in Table 1 that the first two columns present the instance characteristics in terms of number of vessels and number of ports. In the remaining columns we have considered several performance measures. In the third column the objective function values of the optimal integer solution are depicted. Column four illustrates the computational time to solve the integer optimal solution. In column five the integrality GAP is shown. The integrality GAP is determined through the expression: $GAP = (Z^*_{pi} - Z^*_{rl}) / Z^*_{pi}$, where Z^*_{pi} denotes the value of the optimal integer solution and Z^*_{rl} is associated with the linear relaxation optimal solution. Aiming at giving an idea of the MIP model dimension, in terms of the number of variables and constraints, in the last two columns we present its values for the set of instances solved. Except for the problem with 10 ports and 2 vessels the model always achieves the optimal integer solution without overstows. As could be seen in Table 1, after 15 minutes running in CPLEX, the model had not yet come to a solution. So, we will consider that this

problem hasn't achieved an optimal solution in a reasonable computational time [15]. The computational time increases with the number of vessels and the number of ports. For the problems with one vessel, increasing the number of ports results in a small increasing of the computation time. However there is an exponential rise of the computational time when, for the same number of ports the number of vessels is increased. This was expected due to the number of binary variables related to the containers' positioning. Taking into account the set of tested problems we can conclude that the integrality GAP decreases with the increase of the number of ports. The opposite behavior happens when varying the number of vessels. Nevertheless, with these results it is proven that this model can be applied to real problems due to the small computational time needed to achieve an optimal solution.

SA2: Varying the demand distribution per port:
The behavior of the model and the effect of varying the demand distribution per port was also explored (Table 2). As in Moura et al. [3], three types of demands' distribution were considered: homogeneous, weakly heterogeneous and strongly heterogeneous. As explained before, the homogeneous load distribution happens when the containers number and weight is almost equal for all ports. On the other hand, it is weakly heterogeneous when there are some differences between them. Furthermore, strongly heterogeneous is when differences between the demands of each port are significant.

SA3: Varying the deadlines:
Another test was made in order to see the behavior of the model according to different types of deadlines. In this case, narrow and wide deadlines were tested using the homogenous load distribution problems with two vessels for five ports (table 2). When a deadline is small, it is called a narrow deadline, on the other hand when a deadline is big it is called a wide deadline.

			Value Integer Optimal Solution (€)	Computational Time (seconds)	GAP (%)
5 Ports 2 Vessels	Demands	Strongly heterogeneous	237706,61	7,92	91,32
		Homogeneous	239637,75	94,43	89,7
		Weakly heterogeneous	201606,94	40,61	89,78
	Deadlines	Narrow	239637,75	30,53	89,71
		Normal	239637,75	94,43	89,7
		Wide	239637,75	83,99	89,71

Table 2 - Computational results varing the demands and deadlines

The effects of varying the demands distribution and deadlines in the solution computational times are in Table 2. These two data types are critical because, for example, if the load to a given port is equal to or very close to the vessel capacity, that vessel will visit only that port, making the problem potentially easier to solve due to the decrease in the number of possible route combinations. The results obtained for five ports (Table 2) and also proven with ten ports, denote a relationship between the type of demand and the computational time needed to solve the integer problem. For strongly heterogeneous distribution the computational time is always smaller than considering the other two load distributions. Moreover, the highest computational time is always associated with the homogeneous distribution of the cargo. Also in Table 2 the effects of the variation of the deadlines (all combined with homogeneous cargo distribution) in the solution computational times are presented. With wide deadlines the computational time is similar to the computational time of regular deadlines. But decreasing the size of the deadlines also decreases the computational time, and in a significant way. We think that this fact can be explained with the same arguments presented above for the demands variation.

5 Conclusions

In this work a Mixed Linear Programing model to the short sea distribution problem defined has CSSRP, was presented. This problem can be approached like the integration of two well-known NP-Hard problems: the Vehicle Routing Problem and the 3-Dimentional Loading Problem. However in CSSRP the demands have deadlines and the containers placement locations are fixed. Besides, in CSSRP it must be decided where to load the demands to a given port in order to reduce the unloading time and prevent overstowing.
The CSSRP is also a NP-hard problem. Despite the complexity of the problem and of the presented model, the results obtained for the set of problems tested prove that this model can be applied to solve real life short sea shipping problems. It was proven that the model always reaches an optimal integer solution, without unnecessary movements of cargo during the trips, within a very short computational time. In addition, the results showed that the computational time to get the optimal solution to the problem besides depending on number of ports to visit and number of vessels also depends on the type of demands distribution and deadlines.
The developed approach in this paper shows that the solutions quality of the short sea shipping problem can be improved by having an integrated perspective of the two above mentioned problems (the VRP problem and the CSP problem).

References

1. A. MOURA AND J.F. OLIVEIRA, An integrated approach to the vehicle routing and container loading problems, Operations Research Spectrum 31, 775-800, (2009).
2. M. GENDREAU, M. IORI, G. LAPORTE AND S. MARTELLO, A tabu search algorithm for a routing and container loading problem, Transportation Science 9(3), 342-350, (2006).
3. A. MOURA, J. OLIVEIRA AND C. PIMENTEL, “A Mathematical Model for the Container Stowage and Ship Routing Problem”, Journal of Mathematical Modeling and Algorithms, pp. 1-15; DOI: 10.1007/s10852-012-9207-3, (2012).
4. I.D. WILSON AND P.A. ROACH, Principles of combinatorial optimization applied to container-ship stowage planning, Journal of Heuristics 5(4), 403-418(16), (1999).
5. M. CHRISTIANSEN AND B. NYGREEN, A method for solving ship routing problems with inventory constraints, Annals of Operational Research 81, 357-378, (1998).
6. R. AGARWAL AND O. ERGUN, Ship scheduling and network design for cargo routing in linear shipping, Transportation Science 42(2), 175-196, (2008).
7. T. MARTINS, A. MOURA, A.A. CAMPOS AND V. LOBO, Genetic algorithms approach for containerships fleet management dependent on cargo and their deadlines, IAME 2010: Annual Conference of the International Association of Maritime Economists Lisbon 7-9 July (2010).
8. M. AVRIEL, M. PENN AND N. SHPIRER, Container ship stowage problem: complexity and connection to the coloring of circle graphs, Discrete Applied Mathematics 103, 271-279, (2000).
9. A.H. ASLIDIS, Combinatorial algorithms for stacking problems, in PhD dissertation, MIT, (2000).
10. M. AVRIEL AND M. PENN, Exact and approximate solutions of the container ship stowage problem, Computers and Industrial Engineering 25, 271–274, (1993).
11. M. AVRIEL, M. PENN, N. SHPIRER AND S. WITTEBOON, Stowage planning for container ships to reduce the number of shifts, Annals of Operations Research 76, 55-71, (1998).
12. R.C. BOTTER AND, M.A. BRINATI, Stowage container planning: a model for getting an optimal solution, Computer Applications in the Automation of Shipyard Operation and Ship Design, VII. North-Holland, 217–229, (1992).

13. D. AMBROSINO, A. SCIOMACHEN AND E. TANFANI, Stowing a containership: the master bay plan problem, Transportation Research Part A 38, 81-99, (2004).
14. D. AMBROSINO, A. SCIOMACHEN AND E. TANFANI, A decomposition heuristics for the container ship stowage problem, Journal of Heuristics 12, 211–233, (2006).
15. A. DELGADO, R.M. JENSEN, K. JANSTRUP, T.H. ROSE AND K.H. ANDERSEN, A Constraint Programming model for fast optimal stowage of container vessel bays, European Journal of Operational Research 220, 251-261, (2012).
16. O. DUBROVSKY, O.G. LEVITIN AND M. PENN, A genetic algorithm with a compact solution encoding for the container ship stowage problem, Journal of Heuristics 8, 585-599, (2002).
17. I.D. WILSON AND P.A. ROACH, Container stowage planning: a methodology for generating computerized solutions, Journal of the Operational Research Society 51, 1248-1255, (2000).
18. Z. WEI-YING, L. YAN AND J.T. ZHUO-SHANG, Model and algorithm for container ship stowage planning based on Bin-packing problem, Journal of Marine Science and Application 4(3), (2005).

MISTA 2013

Solving a cooperative project scheduling with controllable processing times, self-interested agents and equal profit sharing

Cyril Briand, Sandra Ulrich Ngueveu and Přemysl Šůcha

Abstract This paper considers a cooperative project that involves a set of self-interested agents, each in charge of a part of the project. An agent can reduce the duration of the activities he is responsible for, by gathering extra-resources, at a given cost. If the overall project ends earlier than expected, the client offers a reward to be shared equally among agents. The financial outcome for each agent therefore depends on all agents decisions and on the satisfaction of the customer, which is function of the project makespan. Under these assumptions, we address the problem of finding a Nash equilibrium, ensuring the stability of the schedule, that minimizes the project makespan. We explain how this problem, which is NP-hard, can be efficiently modeled and solved with mixed integer linear programming.

1 Introduction

Large-size projects in the building and construction sector or automotive industries usually require cooperation among groups of self-interested actors, often contractors, each being in charge of a part of the project, as described in [6] or [7]. These actors, often referred as *agents* in the literature, control the duration of activities they are responsible for. They can reduce durations by gathering extra-resources, but at a given cost. The client of the project is interested in minimizing the total duration of his project, and as an incentive, offers a daily reward to be shared among agents and

Cyril Briand
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France and
Univ de Toulouse, UPS, F-31400 Toulouse, France
E-mail: briand@laas.fr

Sandra Ulrich Ngueveu
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France and
Univ de Toulouse, INP, F-31400 Toulouse, France
E-mail: ngueveu@laas.fr

Přemysl Šůcha
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France and
Czech Technical University in Prague; Faculty of Electrical Engineering; Karlovo namesti 13, 121 35, Prague 2, Czech Republic
E-mail: suchap@fel.cvut.cz

proportional to the earliness of the realization [4]. Let us note that this notion of reward is symmetric to the one of penalty that agents must pay proportionally to the tardiness. Therefore, the payoff for each agent depends not only on his own decision, but also on the decisions of all other agents, which translate into a given project makespan, and thus a given reward.

Under these assumptions, we address the problem of finding a Nash equilibrium that minimizes the project makespan, assuming that the reward is equally shared among the agents involved into the project. In other words, we aim at determining a strategy that favor the customer objective with the constraint that the strategy remains stable (no agent will have interest in changing his decisions). This problem has been already considered in the literature by Agnetis et al. [1] and proved to be NP-Hard. This paper shows how such a problem can be modeled and solved efficiently with mixed integer linear programming (MILP).

The paper is structured as follows. Section 2-4 give a formal definition of our problem and recall some basic concepts and properties. Section 5 proposes an original MILP formulation for that problem. Section 6 reports some computational experiments.

2 Problem statement

The multi-agent project scheduling problem considered in this paper can be modeled with an activity-on-arc graph $\mathcal{G} = (X, U)$ that defines the project activity network. The set of nodes $X = \{0, 1, \ldots, n-1\}$ represents the project events, and the set of arcs U represents project activities. Nodes 0 and $(n-1)$ correspond to the project beginning and project end. The set of arcs U results from the union of two subsets $U_R \cup U_D$, where U_R is the set of real project activities and U_D is the set of dummy activities. Dummy activities are related to precedence constraints. U_R is divided among a set $\mathcal{A} = \{A_1, \ldots, A_m\}$ of m agents.

The set of m_u activities assigned to agent A_u is denoted $\mathcal{T}_u$. $\mathcal{T}_u \cap \mathcal{T}_v = \emptyset$ for any pair of agents $(A_u, A_v) \in \mathcal{A}^2$ such that $u \neq v$. Each activity $(i,j) \in U$ is associated with a minimal duration $\underline{p}_{i,j}$ and a normal duration $\overline{p}_{i,j}$.

For each activity $(i,j) \in \mathcal{T}_u$, the responsible agent A_u can choose a processing time $p_{i,j} \in [\underline{p}_{i,j}, \overline{p}_{i,j}]$. For each dummy activity $(i,j) \in U_D$, $\underline{p}_{i,j} = \overline{p}_{i,j} = 0$. A crashing unitary-cost $c_{i,j}$ is associated to each activity $(i,j) \in U$ (provided that $c_{i,j} = 0, \forall (i,j) \in U_D$). The cost incurred by an agent A_u when shortening an activity $(i,j) \in \mathcal{T}_u$ from $\overline{p}_{i,j}$ to $p_{i,j}$ is therefore $(\overline{p}_{i,j} - p_{i,j})c_{i,j}$.

The strategy of an agent A_u is denoted P_u and is the duration vector of all his activities $\mathcal{T}_u$. All individual strategies can be gathered into a vector $S = (P_1, \ldots, P_m)$ called strategy profile. For any given strategy profile S, the resulting project makespan $D(S)$ can be computed using a classical longest path algorithm on the graph $\mathcal{G}(S)$, which results from $\mathcal{G}$ by setting the length of every arc to its corresponding value in strategy profile S. We further refer to $\overline{D} = D(\overline{S})$ as the project makespan obtained when every activity duration is set to its normal (and thus maximal) value $\overline{p}_{i,j}$.

Let parameter π be the daily reward offered by the client, therefore the value $Z_u(S) = \frac{\pi}{m}(\overline{D} - D(S)) - \sum_{(i,j)\in\mathcal{T}_u} c_{i,j}(\overline{p}_{i,j} - p_{i,j})$ corresponds to the profit of A_u for the strategy profile S, where $(\overline{D} - D(S))$ is the project-makespan reduction. This profit is equal to the difference between the reward received from the client and the expenses incurred to shorten tasks durations.

3 Efficiency vs. stability

A strategy profile is said *efficient* if it corresponds to a Pareto-optimal solution. In other word, if no other strategy profile S' exists that dominates S with respect to the agents' profit. On the other hand, a strategy profile is *stable* if there is no incentive for any agent to modify it in order to improve his profit. The stability of a strategy profile is important since it ensures that agents can trust each other. It is totally connected to the notion of a Nash equilibrium.

Let S_{-u} denote the strategies played by the $(m-1)$ agents except A_u, (*i.e.*, $S_{-u} = (P_1, \ldots, P_{u-1}, P_{u+1}, \ldots P_m)$). Then, focusing on a particular agent A_u, any S can be also written as a couple (P_u, S_{-u}). A strategy profile vector $S = (P_u, S_{-u})$ is a Nash equilibrium if for all agents A_u and any local alternate strategy P'_u, we have $Z_u(P_u, S_{-u}) \geq Z_u(P'_u, S_{-u})$.

For illustration, let us consider the activity network of Figure 1 with 5 activities distributed among the two agents A_1 and A_2 such that $\mathcal{T}_1 = \{a, c\}$ and $\mathcal{T}_2 = \{b, d, e\}$. We assume that the daily reward π equals 120 to be shared fairly (*i.e.*, $w_1 = w_2 = 1/2$). When the durations of all activities are set to their maximal value, *i.e.*, $S = (\overline{p}_a, \overline{p}_b, \overline{p}_c, \overline{p}_d, \overline{p}_e) = (7, 9, 3, 8, 5)$, the project makespan is 15 and no agent makes any profit ($Z_1(S) = Z_2(S) = 0$). With the strategy profile $S' = (7, 9, \mathbf{2}, \mathbf{7}, 5)$, the makespan becomes 14 and profits are $Z_1(S') = Z_2(S') = 40$. The makespan can be further reduced to 13 with the strategy profile $S'' = (\mathbf{6}, 9, \mathbf{3}, 7, \mathbf{4})$, leading to $Z_1(S'') = 50$ and $Z_2(S'') = 50$. S'' is efficient since it corresponds to a Pareto optimum that maximizes the profit of both agents. Nevertheless, it is not stable: agent A_2 can improve his profit, to the detriment of A_1, by simply increasing back the duration of d and e, which leads to the strategy profile $S''' = (6, 9, 3, \mathbf{8}, \mathbf{5})$, with makespan 14 and profits $Z_2(S''') = 60$ and $Z_1(S''') = -10$. On the other hand, the strategy profile S', which is not a Pareto optimum, is stable since no agent is able to improve his profit by himself, *i.e.*, it is a Nash equilibrium.

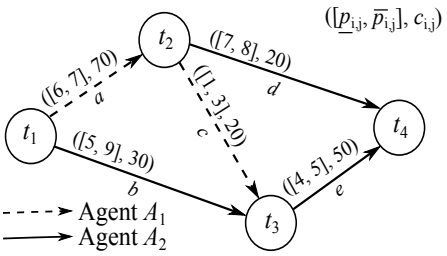


Fig. 1 A multiagent activity network with two agents and five activities

To sum up, the notion of Pareto strategy is important in regards of the agents profits. The notion of Nash equilibrium is complementary to the previous one and ensures organizational stability. Ideally, agents should choose a strategy that is both a Pareto optimum and a Nash equilibrium, but such a strategy does not always exist (as shown for the example in Figure 1).

In addition, from the customer viewpoint, the problem of finding a Nash equilibrium that minimizes the project makespan is also of interest: its solution gives the lower

makespan that can be reached for the customer, provided that the organization remains stable. It can be viewed as the problem of minimizing the price of stability from the customer viewpoint [3]. We recall that such problem has been proved NP-hard in [1] using a reduction from problem 3-PARTITION.

Let us also point out that, when only a single agent is considered (instead of multiple agents), our problem reduces to the well-known discrete time-cost trade-off problem in project scheduling (that can be solved in polynomial time [8]).

4 Characterization of a Nash equilibrium

As proposed by Agnetis et al., a Nash equilibrium can be characterized using the notion of residual cuts in critical activity networks. This notion of residual cut is also used in [8] for solving the time-cost trade-off in project scheduling problem (single agent case), and in [5] for increasing or decreasing the tension in a graph at minimum cost.

Given a cut $\omega = (\Omega, \overline{\Omega})$ (with $0 \in \Omega$ and $n-1 \in \overline{\Omega}$), let us refer to ω^+ (resp. to ω^-) as the set of critical activities (i,j) such that $i \in \Omega$ and $j \in \overline{\Omega}$ (resp such that $i \in \overline{\Omega}$ and $j \in \Omega$). A cut ω is said residual with respect to a strategy profile S if there exists $\delta \in \mathbb{R}$ such that:

$$\begin{cases} p_{i,j} + \delta \in [\underline{p}_{i,j}, \overline{p}_{i,j}], \forall (i,j) \in \omega^+ \\ p_{i,j} - \delta \in [\underline{p}_{i,j}, \overline{p}_{i,j}], \forall (i,j) \in \omega^- \end{cases} \quad (1)$$

An increasing cut identifies a way to increase by $\delta > 0$ the project makespan with the unitary profit $Profit(\omega) = +(\sum_{(i,j)\in\omega^+} c_{i,j} - \sum_{(i,j)\in\omega^-} c_{i,j})$. Similarly, a decreasing-cut allows to decrease by $\delta < 0$ the project makespan with the unitary profit $Profit(\omega) = -(\sum_{(i,j)\in\omega^+} c_{i,j} - \sum_{(i,j)\in\omega^-} c_{i,j})$. When only the activities belonging to one agent A_u are taken into account in the previous formula, we obtain the projected cost, $Profit_u(\omega)$, which gives the unitary profit for agent A_u with respect to cut ω. It corresponds to the increase or decrease of the profit of agent A_u when modifying the duration of the activities involved in ω.

We also need to introduce the notion of a *poor* strategy profile. A strategy profile S with project duration $D(S)$ is said poor if there exists an alternative strategy profile $S' = (P'_u, S_{-u})$ with the same makespan $D(S') = D(S)$, only differing from S by the strategy taken by A_u, such that $Z_u(S) < Z_u(S')$ and $Z_v(S) \leq Z_v(S')$, for all $A_v \neq A_u$. In other words, given the strategies chosen by all agent except A_u, a poor strategy profile will not maximize the profit of agent A_u. Obviously, any poor strategy profile can be neither a Pareto or a Nash one.

With that in mind, the following proposition holds [1], aiming at characterizing a Nash equilibrium.

Proposition 1 *A non-poor strategy profile S is a Nash equilibrium if and only if there is no residual cut $\omega(\Omega, \overline{\Omega})$ satisfying one of the two following conditions: (1) $\exists \delta > 0$ and $\exists A_u \in \mathcal{A}$ with $Profit_u(\omega) \geq \frac{\pi}{m}$ or (2) $\exists \delta < 0$ and $\forall (i,j) \in \omega^+$, $(i,j) \in \mathcal{T}_u$ and $-Profit_u(\omega) > \frac{\pi}{m}$.*

The proof of this proposition is summarized below. First, S being a non-poor strategy profile, the only way for an agent to increase his profit is to increase or decrease the project makespan, *i.e.*, to identify either an increasing or decreasing cut being profitable for him. Condition (1) ensures that there cannot exist any increasing cut

profitable for any agent, *i.e.*, such that the agent's profit is greater than his part of daily reward. Similarly, Condition (2) states that there cannot exist any decreasing cut profitable for one agent, *i.e.*, such that the daily reward part is greater than the agent's profit. If both conditions (1-2) stand, then no agent can individually make profit and therefore strategy profile S is a Nash equilibrium.

5 Finding a Nash equilibrium S minimizing $D(S)$

According to Proposition 1, the problem of finding a Nash equilibrium that minimizes the makespan can be formulated as a mathematical program as follows.

$$\begin{array}{lll} \min & t_{n-1} - t_0 & (1) \\ \text{s.t} & t_j - t_i - p_{i,j} \geq 0 \quad ,\forall (i,j) \in U & (2) \\ & \underline{p}_{i,j} \leq p_{i,j} \leq \overline{p}_{i,j} \quad ,\forall (i,j) \in U & (3) \\ & Profit_u(\omega) < \frac{\pi}{m} \quad \forall \text{ cut } \omega, \forall A_u \in \mathcal{A} & (4) \\ & t_i \in \mathbb{R} \quad ,\forall i \in X & (5) \end{array}$$

Variables t_i, with $i \in [0, n-1]$, are associated with occurrence times of the project events of X. The objective function (1) aims at minimizing the project makespan. Let us note that this objective ensures that the optimal strategy profile cannot admit any decreasing cut, so that the condition (2) of Proposition 1 is enforced. Constraints (2) model precedence relations between project activities, constraints (3) specify the bounds on the processing times values. With respect to condition (1) of Proposition 1, the high-level constraints (4) impose that any existing residual cut must not have a projected profit greater or equal to $w_u\pi$. Because the number of cuts in $\mathcal{G}$ is exponential, the resulting problem remains hard to handle. However, as explained in the sequel, we were able to model constraints (4) efficiently inside the MILP, using a finite number of primal-dual constraints.

First, it was noted that, for each agent, determining a cut that maximizes $Profit_u(\omega)$ facilitates the verification of constraints (4). Such problem is a max-cut problem that can be formulated as the linear program (6)-(10). Variable $\gamma_i^u = 1$ if i is in Ω, and 0 otherwise. Variable $\alpha_{i,j}^u = 1$ if $(i,j) \in \omega^+$, and 0 otherwise. Similarly, $\beta_{i,j}^u = 1$ if $(i,j) \in \omega^-$, and 0 otherwise. The parameters $[\underline{c}_{i,j}^u, \overline{c}_{i,j}^u]$ are the minimum and maximum capacity of each arc (i,j). If (i,j) is in $\mathcal{T}_u$ and is critical and $p_{i,j} > \underline{p}_{i,j}$, then $\overline{c}_{i,j}^u = c_{i,j}$. If $(i,j) \notin \mathcal{T}_u$ and is not critical, then $\overline{c}_{i,j}^u = 0$. For any other case, $\overline{c}_{i,j}^u = \infty$. If (i,j) is in $\mathcal{T}_u$ and is critical and $p_{i,j} < \overline{p}_{i,j}$, then $\underline{c}_{i,j}^u = c_{i,j}$. For any other case, $\underline{c}_{i,j}^u = 0$.

$$\begin{array}{lll} \max & \sum_{(i,j)\in\mathcal{T}_u} \alpha_{i,j}^u \underline{c}_{i,j}^u - \sum_{(i,j)\in\mathcal{T}_u} \beta_{i,j}^u \overline{c}_{i,j}^u & (6) \\ \text{s.t} & \alpha_{i,j}^u - \beta_{i,j}^u - \gamma_i^u + \gamma_j^u \leq 0 \quad ,\forall (i,j) \in U & (7) \\ & \gamma_0^u = 1, \gamma_{n-1}^u = 0 & (8) \\ & \gamma_i^u \geq 0 \quad ,\forall i \in X & (9) \\ & \alpha_{i,j}^u, \beta_{i,j}^u \geq 0 \quad ,\forall (i,j) \in U & (10) \end{array}$$

The dual of this problem is a min-flow problem, that can also be expressed as the linear program (11)-(14) where $\phi_{i,j}^u$ is the flow assigned to A_u and circulating on $(i,j) \in U$:

$$\begin{array}{lll} \min & \sum_{k\in X\setminus\{0\}} \phi^u_{0,k} & (11) \\ \text{s.t} & \sum_{(k,i)\in U} \phi^u_{k,i} = \sum_{(i,k)\in U} \phi^u_{i,k} \quad ,\forall i \in X & (12) \\ & \underline{c}^u_{i,j} \leq \phi^u_{i,j} \leq \overline{c}^u_{i,j} \quad ,\forall (i,j) \in U & (13) \\ & \phi^u_{i,j} \geq 0 \quad ,\forall (i,j) \in U & (14) \end{array}$$

From the duality theory, we know that only an optimal solution can satisfy both sets of constraints (7)-(10) and (12)-(14) as well as additional constraint: $\sum_{k\in X\setminus\{0\}} \phi^u_{0,k} \leq \sum_{(i,j)\in\mathcal{T}_u} \alpha^u_{i,j}\underline{c}^u_{i,j} - \sum_{(i,j)\in\mathcal{T}_u} \beta^u_{i,j}\overline{c}^u_{i,j}$ (15) (which imposes that primal and dual objective functions are equal).

To summarize, in order to find a minimum makespan Nash equilibrium, we propose the MILP obtained from the model (1)-(5) after replacing constraints (4) by constraints (7)-(10) and (11)-(15), with the addition of constraints: $\sum_{k\in[1,n-1]} \phi^u_{0,k} < \frac{\pi}{m}$ (16).

Note that $\underline{c}^u_{i,j}$ and $\overline{c}^u_{i,j}$ depend on the $p_{i,j}$ variables, therefore the constraint (15) is not linear because it involves multiplication of variables. Nevertheless, it can be linearized efficiently because each multiplication involves one binary and one real variable.

6 Preliminary numerical results

The resulting MILP model was implemented in IBM ILOG CPLEX Optimization Studio 12.3 and solved with the CPLEX solver on a PC with 8 GB of RAM and 3.0 GHz CPU. It was applied on the 4 largest set of instances $|U_R| = \{20, 40, 60, 80, 100\}$ available from authors upon request. Each benchmark set consists of 100 problem instances considering 5 agents + 1 virtual agent for dummy activities. The daily reward π is determined with respect to the maximal cut cost $C(\omega_{max})$, which represents the theoretical most expensive cut that can be used to decrease project makespan when affiliations to agents are ignored.

The results in terms of computing times for $\pi = 0.05 \cdot C(\omega_{max})$ are available in Table 1. It shows the average, the minimal and the maximal CPU time needed to solve the instances for each $|U_R|$. In addition, the table also provides the total number of activities U and nodes X from the network graphs corresponding to the instances.

$\|U_R\|$	avg $\|U\|$	avg $\|X\|$	CPU time		
			avg	min	max
[–]	[–]	[–]	[s]	[s]	[s]
20	50.6	20.3	0.27	0.02	0.92
40	162.4	47.2	0.41	0.05	1.56
60	327.4	76.4	1.17	0.09	3.49
80	538.4	106.2	2.00	0.78	4.60
100	790.9	135.0	3.16	1.30	16.35

Table 1 Resolution time for our MILP depending on the number of activities

Analysis of this table shows that (i) the total number of activities is significantly larger than the number of real activities, and (ii) the solving phase remains very fast since we are able to solve instances with 100 real activities (almost 800 total activities!) in less than few seconds on average.

The daily reward π is an important parameter for the project client. Of course, it can be expected that the larger π, the shorter the project makespan accessible.

However, no client wants to waste money by paying more than what is necessary to reach a given makespan, or worse, by paying more without any makespan improvement. Figure 2 shows the project makespan gain $D^*/\overline{D}$ in function of the customer reward π obtained on the benchmark set with $|U_R| = 60$.

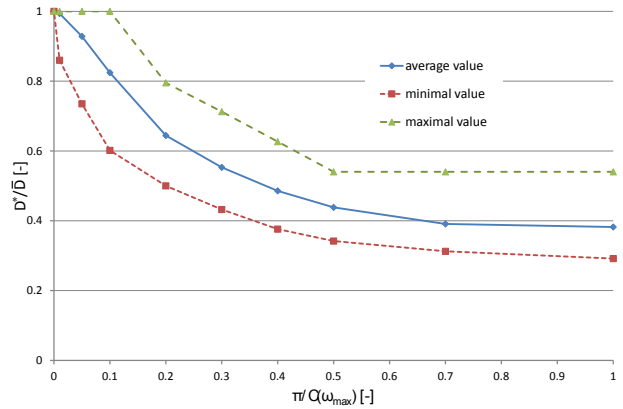


Fig. 2 Project makespan gain in function of the customer reward

As expected, the makespan decreases when the reward increases. However, it is important to note that (i) the correlation is not linear, (ii) until approximately $0.3C(\omega_{max})$ every additional reward results into significant makespan decrease, but after $0.3C(\omega_{max})$, each additional reward leads to less and less improvement to the makespan until $0.7C(\omega_{max})$, after which almost no improvement is obtained, and of course, (iii) there is a makespan lower bound that cannot be overpassed.

A project customer should therefore apply our model on the specific instance corresponding to his own project, in order to identify the best way to set its reward that would lead to the makespan reduction he aims at.

7 Conclusion

This paper presents a MILP formulation based on the characterization of a Nash equilibrium with residual cuts in a critical activity networks, to model a multi-agent cooperative project with self-interested agents and equal profit-sharing. The model obtained has been implemented and tested on several problem instances. Preliminary results are encouraging since optimal solutions are found in a few seconds for problems having up to 100 real activities (800 total activities) and 6 agents. Large scale experiments are in progress and a deeper analysis of the performance of our approach will be provided during the conference. Another advantage of the MILP formulation introduced here lies in the fact that the profit share of each agent may be replaced by a parameter so that any other predefined sharing policy may be implementable, without changing the nature of the MILP. Such a study is of interest, especially if the customer is able to influence the rewards sharing policy, because it could help designing more efficient sharing policies for the problem considered.

Acknowledgements This work was supported by the ANR project no. 08-BLAN-0331-02 named "ROBOCOOP".

References

1. Agnetis, A., Briand, C., and Billaut, J.-C, Přemysl Š. 2012. Nash equilibria for the multi-agent project scheduling problem with controllable processing times, Research Report, LAAS-CNRS, http://homepages.laas.fr/briand/mapsp3.pdf.
2. Cachon, G. P. and Lariviere, M. A. 2005. Supply chain coordination with revenue-sharing contracts: Strengths and limitations. *Management Science* 51(1): 30–44.
3. Christodoulou, G. and Koutsoupias, E. 2005. The price of anarchy of finite congestion games, *37th Annual ACM STOC*, 67–73.
4. Fernandez, E. 2008. A game theoretical approach to sharing penalties and rewards in projects, European Journal of Operational Research 01; 216:647-657.
5. Hadjiat, M. and Maurras, J. F. 1997. A strongly polynomial algorithm for the minimum cost tension problem. *Discrete Mathematics* 165-166: 377–394.
6. Leufkens , A. S. and Noorderhaven, N. G. 2011. Learning to collaborate in multi-organizational projects. *International Journal of Project Management* 29(4): 432–441.
7. Martin Diaz, L. 2006. Evaluation of cooperative planning in supply chains : an empirical approach of the european automotive industry. In Publications of Darmstadt Technical University, Institute for Business Studies (BWL) 25825, Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies.
8. Philips S.J. and Dessouky, M. I. 1977. Solving the project time/cost tradeoff problem using the minimal cut concept. *Management Science* 24(4): 393–400.

MISTA 2013

Production and Interplant Batch Delivery Scheduling: Dominance and Cooperation

Alessandro Agnetis · Mohamed Ali Aloulou · Liang-Liang Fu

Abstract In this paper, we consider scheduling problem in a supply chain with two actors, a manufacturer and a third-party logistics(3PL). The manufacturer has to process a set of jobs at the upstream stage and the downstream stage. The 3PL is in charge of transportation of semi-finished products from the upstream stage to the downstream stage. The manufacturer's objective is to minimize makespan $C_{\max}$ and the 3PL's objective is to minimize transportation cost TC. We investigate three different types of contract: the manufacturer dominates, the 3PL adjusts; the manufacturer dominates, the 3PL negotiates; the manufacturer and the 3PL have the same bargaining power and cooperate. We provide exact polynomial-time algorithms or prove the NP-completeness of the scheduling problems in these three types of contract. We evaluate the benefit of cooperation using numerical experiments.

1 Introduction

Supply chain management is an active domain consisting of the optimization and management of flows between different actors in the chain, such as suppliers, manufacturers, logistics providers, distributors and customers. These actors generally have conflicting objectives, which makes the coordination of their decisions a crucial issue in supply chain management.

In recent years, the supply chain coordination issue has received great interest from several researchers. Before 2000, most of the work has focused on coordination at the strategic and tactical levels, see e.g. the surveys by Sarmiento and Nagi [15] and Erenguc et al. [7]. Thomas and Griffin [16] pointed out the need for research

Alessandro Agnetis
Università di Siena, via Roma 56, 53100 Siena, Italy
E-mail: agnetis@dii.unisi.it

Mohamed Ali Aloulou
Université Paris-Dauphine, LAMSADE - CNRS, Paris, France
E-mail: Mohamed-Ali.ALOULOU@dauphine.fr

Liang-Liang Fu
Université Paris-Dauphine, LAMSADE - CNRS, Paris, France
E-mail: liangliang.fu@dauphine.fr

that addresses supply chain issues at an operational level rather than a strategic level. This has triggered a certain amount of research on supply chain coordination at the operational level.

Hall and Potts [10] provide the first study of supply chain scheduling. They consider the coordination between scheduling, batching and delivery decisions in a 3 stages supply chain. They study two individual decision models and the integrated model, and evaluate the benefit of cooperation. Agnetis et al. [1] study the coordinated scheduling problem between a supplier and several manufacturers with consideration of an intermediate storage buffer. They provide algorithms for two individual decision problems and the integrated problem with only one manufacturer. There are several articles developing the supply chain scheduling with cost of conflict, mechanism of cooperation in different bargaining power scenarios (e.g. Dawande et al. [6], Chen and Hall [5]) and cooperative sequencing game (e.g. Aydinliyim and Vairaktarakis [3]).

Recently, a growing amount of papers have been devoted to studying integrated supply chain scheduling issues. Chen [4] surveys integrated production and outbound distribution scheduling. Most of the vast literature on *outbound* distribution deals with a manufacturer shipping its products to the next stage of the supply chain, that typically belongs to another subject. As a consequence, the receiving firm may set due dates or deadlines that will constrain the production/distribution problem. Here we are concerned with the situation arising when the same manufacturer owns both the upstream and the downstream stage. This case is referred to as *interplant* distribution. In this case, transportation can be outsourced to a third-party logistics (3PL). In the literature, there are not many articles about supply chain scheduling problems with the logistics provider (3PL) (Li et al. [14], Zhong et al. [17]).

In the literature, a few articles address integrated production and interplant distribution scheduling issues. Lee and Chen [12] study a 2-machine flow shop makespan minimization problem with the consideration of capacity and transportation time. Lee and Strusevich [13] provide approximation algorithms for the 2-machine flow shop and open shop scheduling problems with a single vehicle and unbounded capacity. Aloulou et al. [2] consider a bicriteria 2-machine flow shop scheduling problem with infinite number of vehicles and limited capacity. They minimize two criteria: the transportation cost and the makespan. They develop two polynomial-time approximation algorithms and provide exact polynomial-time algorithms for some particular cases. These models generalize classical machine scheduling models, and are suitable for a scenario in which the owner of the two facilities takes care of transportation, so that this can be synchronized with processing at the two stages.

In this paper, we consider scheduling problem in a supply chain with two actors, a manufacturer and a 3PL. The manufacturer has to process a set of jobs at the upstream stage and the downstream stage. The 3PL is in charge of transportation of semi-finished product from the upstream stage to the downstream stage. The manufacturer's objective is to minimize makespan $C_{\max}$ and the 3PL's objective is to minimize transportation cost TC.

We are going to investigate three different types of contract.

In the first type, the manufacturer decides the schedule of the jobs that are being released at the upstream stage. The contract requires the 3PL to deliver each of them to the downstream stage, within a certain time T from its release at the upstream stage. The 3PL is therefore left with the problem of minimizing its costs while complying with the responsiveness requirement. We say that in this scenario *manufacturer dominates, 3PL adjusts.*

In the second type of contract, the manufacturer decides also the schedule of the jobs that are being released at the upstream stage and prescribes a certain value of T. This time-lag T induces an upper bound for makespan. The 3PL is allowed to negotiate with the manufacturer to relax the responsiveness requirement. However he has to guarantee the upper bound for makespan. We say that in this scenario *manufacturer dominates, 3PL negotiates*.

In the third type of contract, the two actors have the same bargaining power and decide together a production-distribution schedule, where the cost for each actor is at most the cost he would have without cooperation. We do not consider the responsiveness requirement. In this case, we say that *manufacturer and 3PL cooperate*.

In his survey on integrated production and distribution scheduling problems, Chen [4] observes that research has not yet adequately addressed problems with *fixed delivery departure dates*. In this paper we consider a general setting in which transportation can be alternatively provided by means of resources with fixed departure times as well as dedicated resources which are always available, and deciding which resource is the most appropriate to perform each delivery is part of the 3PL decision problem.

This paper is organized as follows. In section 2, we formally describe the problems and introduce notation and terminology. Section 3 is devoted to two scenarios: the manufacturer dominates, the 3PL adjusts; the manufacturer dominates, the 3PL negotiates. Section 4 is devoted to the scenario in which the manufacturer and the 3PL cooperate. In section 5, we evaluate the benefit of cooperation in to scenarios: the manufacturer dominates, the 3PL negotiates; the manufacturer and the 3PL cooperate. Section 6 contains some conclusions and suggestions for future research.

2 Problem definition and notation

In this section, we formally define the problems addressed in this paper. We refer to the upstream and downstream stages of a supply chain as machine M_1 and M_2 respectively.

Context

- A manufacturer has to process a set of n jobs, $1, \ldots, n$
- Each job $j = 1, \ldots, n$ is first processed on machine M_1, with processing time p_j^1, then on machine M_2, with processing time p_j^2,
- M_1 and M_2 are located into two different cities, thus implying that, after processing on M_1, the jobs have to be shipped from M_1 location to M_2 location for further processing.
- A 3PL provider is in charge of transportation of semi-finished product form M_1 to M_2,
- Denote by C_j^i the completion time of job $j = 1, \ldots, n$, on machine $M_i, i = 1, 2$. The 3PL promises to deliver each job j to machine M_2 within time T from its release from machine M_1, i.e. within a deadline $\tilde{d}_j = C_j^1 + T$. We denote by D_j the actual delivery time of job j to M_2.
- The 3PL uses two transportation modes. We define the capacity of a vehicle as the maximum number of jobs a vehicle can carry, i.e. jobs have equal size.
  - Type-1 transportation is *economic transportation*. There are V_1 identical vehicles of capacity $c_1 \leq n$ each of which has a fixed delivery departure date. The transportation time from M_1 to M_2 is denoted by $l \leq T$. The transportation

cost per delivery is denoted by h_1. We assume we have L fixed delivery departure dates $t_1 < \ldots < t_L$, and $v_s \leq n$ vehicles for departure date $t_s, s = 1, \ldots, L$. We denote by $v_{\max}$ the maximum number of economic vehicles in a departure date, i.e. $v_{\max} = \max_{s=1,\ldots,L} v_s$.
  - Type-2 transportation is *express transportation*. There is an unlimited number of identical vehicles of capacity $c_2 \leq n$, each of which can depart at any time. The transportation time from M_1 to M_2 is the same as that of economic transportation. The transportation cost per delivery is denoted by h_2.
- For both transportation modes, jobs in the same delivery form a batch. A batch is available for delivery if all jobs belonging to this batch are completed on machine M_1. Denote by D_B the delivery time of batch B to machine M_2 location.
- The manufacturer wants to minimize the makespan $C_{\max} = \max_{j=1,\ldots,n} C_j^2$.
- The 3PL aims to minimize the transportation cost $TC = h_1 n_1 + h_2 n_2$, where n_1 and n_2 are the number of delivery batches of type-1 and type-2, respectively.

Problem definition In this paper we consider the following problems.

1. **Manufacturer dominates, 3PL adjusts.**
   (a) **Manufacturer's problem.** In this context, the manufacturer searches an optimal production schedule σ on both machines, consisting of the sequence and the starting time of each job. If the manufacturer wants to find a production schedule independent of the 3PL's schedule, he needs to consider the worst situation, in which all jobs are delivered to M_2 location at their deadline exactly. So the manufacturer's problem is a 2 machines flow shop scheduling problem, minimizing $C_{\max}$, with consideration of constant time-lag T.
   (b) **3PL's problem.** In many contexts, the manufacturer may not allow the 3PL to bargain(Chen and Hall [5]). The 3PL aims at finding an optimal distribution schedule, minimizing TC, subject to the release time C_j^1 and the deadline $\tilde{d}_j = C_j^1 + T$ of each job $j, j = 1, \ldots, n$. The distribution schedule consists of the batching scheme and the assignment of a departure date to each batch. The 3PL's problem can be seen as a IPODS (Integrated production and outbound distribution scheduling) problem with a single machine, deadlines and batch delivery to a single customer, in which the sequence on the machine is fixed.
2. **Manufacturer dominates, 3PL negotiates.**
   (a) **Manufacturer's problem.** It is the same as the manufacturer's problem when the manufacturer dominates and the 3PL adjusts.
   (b) **3PL's problem.** We consider the 3PL's problem when he can negotiate with the manufacturer. In order to relax the constraint of deadline, the 3PL can negotiate with the manufacturer to replace the constraint of T by a makespan constraint, i.e. $C_{\max} \leq C_{\max}(\sigma)$, where $C_{\max}(\sigma)$ is the makespan of schedule σ. Therefore, the objective of the 3PL is to find a distribution schedule minimizing TC subject to makespan constraint.
3. **Manufacturer and 3PL cooperate.** The manufacturer and the 3PL have the same bargaining power. The objective is to find the efficient integrated schedules for a 2 machines flow shop scheduling problem on minimizing $C_{\max}$ and TC with consideration of interplant batch deliveries. In order to motivate the cooperation, we select the integrated schedules, where the cost for each actor is at most the cost he would have without cooperation. We do not consider the constraint of T. We consider the permutation problem, i.e. the production schedule has the same sequence on two machines.

Example We consider the following instance. The manufacturer has to process a set of seven jobs, i.e. $n = 7$. Table 2 gives their processing time on the two machines. The 3PL has two types of vehicles, which have the same capacity and the same speed, i.e. $c_1 = c_2 = 3$ and $l = 2$. The transportation cost of economic vehicle equals to one while the transportation cost of express vehicle equals to two, i.e. $h_1 = 1$ and $h_2 = 2$. For economic transportation, there are three fixed departure dates, $t_1 = 13, t_2 = 29$ and $t_3 = 45$. For each fixed departure date, there is one economic vehicle, i.e. $v_s = 1, s = 1, 2, 3$.

Job j	1	2	3	4	5	6	7
p_j^1	1	2	11	16	10	3	2
p_j^2	6	8	10	4	3	2	1

Table 1 Processing time of jobs on the two machines

1. **Manufacturer dominates, 3PL adjusts.**
   (a) **Manufacturer's problem.** There is an optimal production schedule σ generated by Johnson's algorithm (see section 3.1). Johnson's sequence is $\pi^J = \{1, 2, 3, 4, 5, 6, 7\}$. Since $C_{\max}(\sigma) = C_{\max}(\pi^J) + T$, where $C_{\max}(\pi^J) = 46$ and $T = 14$, so $C_{\max}(\sigma) = 60$. The figure 1 illustrates Johnson's sequence.

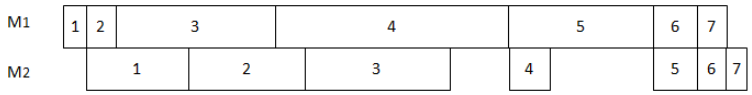


Fig. 1 Illustration of Johnson's sequence

   (b) **3PL's problem.** There is an optimal distribution schedule δ generated by the dynamic programming algorithm of section 3.2.1. There are two economic batches, $\{1, 2\}$, $\{6, 7\}$ which depart at 13 and 45 respectively. There are two express batches, $\{3\}$ and $\{4, 5\}$, which depart at 14 and 40 respectively. Therefore, $TC(\delta) = 2h_1 + 2h_2 = 6$.
2. **Manufacturer dominates, 3PL negotiates.**
   (a) **Manufacturer's problem.** It is the same as the manufacturer's problem when the manufacturer dominates and the 3PL adjusts.
   (b) **3PL's problem.** There is an optimal distribution schedule θ generated by the dynamic programming algorithm of section 3.2.2. There are two economic batches, $\{1, 2\}$ and $\{5, 6, 7\}$, which depart at 13 and 45 respectively. There is one express batch $\{3, 4\}$ which departs at 30. Therefore, $TC(\theta) = 2h_1 + h_2 = 4$. The 3PL proposes a production schedule η corresponding to the distribution schedule θ, where $C_{\max}(\eta) = 53$. The figure 2 illustrates schedule θ and schedule η.
3. **Manufacturer and 3PL cooperate.** There is an efficient integrated schedule λ generated by the mixed integer program of section 4. The sequence of schedule λ is $\{1, 2, 6, 3, 5, 7, 4\}$. There are three economic batches, $\{1, 2, 6\}$, $\{3, 5, 7\}$ and $\{4\}$, which depart at 13, 29 and 45 respectively. Therefore, $TC(\lambda) = 3h_1 = 3$ and $C_{\max}(\lambda) = 51$. The figure 3 illustrates schedule λ.

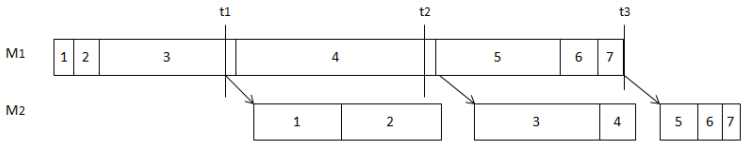


Fig. 2 Illustration of schedule θ and schedule η

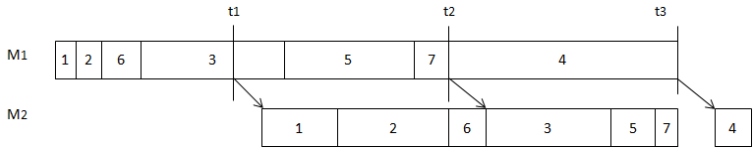


Fig. 3 Illustration of schedule λ

We evaluate the benefit of cooperation in the case where the manufacturer dominates. The 3PL's benefit in this case is defined by $(TC(\delta) - TC(\theta))/TC(\delta)$, which equals to 33%. The manufacturer's benefit in this case is defined by $(C_{\max}(\sigma) - C_{\max}(\eta))/C_{\max}(\sigma)$, which equals to 12%.

We evaluate the benefit of cooperation in the case where the manufacturer and the 3PL have the same bargaining power. The 3PL's benefit in this case is defined by $(TC(\delta) - TC(\lambda))/TC(\delta)$, which equals to 50%. The manufacturer's benefit in this case is defined by $(C_{\max}(\sigma) - C_{\max}(\lambda))/C_{\max}(\sigma)$, which equals to 15%.

3 Manufacturer dominates

3.1 Manufacturer's problem

We consider the optimal schedule σ for the manufacturer's problem. As defined in the previous section, the manufacturer's problem is a 2 machines flow shop scheduling problem, minimizing $C_{\max}$, with consideration of constant time-lag T. Following the commonly used three-field notation $\alpha|\beta|\gamma$ for machine scheduling problems (Graham et al. [9]), this problem is denoted by $F2|time - lag = T|C_{\max}$, which can be solved polynomially by Johnson's algorithm (Johnson [11]). Problem $F2|time - lag = l|C_{\max}$ and problem $F2||C_{\max}$ are equivalent problems. Indeed, we can delay the optimal schedule of problem $F2||C_{\max}$ T time units before machine M_2 to obtain the optimal schedule of problem $F2|time - lag = l|C_{\max}$. Also we can anticipate the optimal schedule of problem $F2|time - lag = l|C_{\max}$ T time units before machine M_2 to obtain the optimal schedule of problem $F2||C_{\max}$. Let π^J be Johnson's sequence. We have,

$$C_{\max}(\sigma) = C_{\max}(\pi^J) + T. \tag{1}$$

3.2 3PL's problems

Before giving solution algorithms for the 3PL's problems, we can prove that the following properties hold.

Lemma 1 *There exists an optimal distribution schedule for the 3PL's problem, such that the following properties hold :*

1. *If job i and job j are delivered by the same transportation mode and $C_i^1 < C_j^1$, the delivery time D_i of job i is not later than the delivery time D_j of job j, i.e. $D_i \leq D_j$.*
2. *Each express batch departs at the completion time of the last job in this batch.*
3. *The jobs of each batch are processed consecutively on machine M_1.*

Proof Before giving the proof, we give the definition of the following notation and action. Denote by $B(i)$ the batch containing job i. Swapping job a and job b in the distribution schedule means that we put job a in the batch containing job b, and job b in the batch containing job a, without changing the production schedule on M_1 and the departure date of each batch.

Property 1: Suppose there is an optimal distribution schedule σ, such that job i is delivered after job j by the same transportation mode and $C_i^1 < C_j^1$. We know $D_{B(j)} < D_{B(i)}$ and $\tilde{d}_i < \tilde{d}_j$. Since $D_{B(i)} \leq \tilde{d}_i$, we have $D_{B(j)} < D_{B(i)} \leq \tilde{d}_i < \tilde{d}_j$. We can swap i and j in the distribution schedule. Applying the same rule, finally we obtain a distribution schedule which respects property 2 and has the same transportation cost as σ. So this new distribution schedule is also optimal.

Property 2: Suppose there is an optimal distribution schedule σ which respects property 1, such that the departure date of an express batch B is later than the completion time of the last job in batch B. We can simply anticipate the departure date of batch B to the completion time of the last job in batch B because there exists an unlimited number of express vehicles.

Property 3: Suppose there is an optimal distribution schedule σ which respects property 1 and property 2, and does not respect property 3.

We assume batch B is the first express batch in schedule σ, in which the jobs are not processed consecutively on machine M_1. According to property 1 and property 2, we known that batch B is only split by economic batches. If batch B is split by another express batch B', then we should have $D_B = D_{B'}$ which we can not realize. We consider a subsequence on machine M_1, denoted by $Se(i, j)$ where i is the first job of batch B and j is the last job of batch B.

Firstly, we search in $Se(i, j)$ the first job u delivered by an economic vehicle, such that $D_u - l \geq C_j^1$. According to property 1, for each job k in an economic batch after job u in $Se(i, j)$, $D_k - l \geq C_j^1$. And for each job w in an economic batch before job u in $Se(i, j)$, $D_w - l < C_j^1$. Secondly, in $Se(i, j)$ before job u, we swap the jobs of batch B with its following job in economic batch, in order that batch B is after the economic batches before job u. In this action, we delay the deadline of the first job of batch B, so the jobs are not late. And we anticipate the deadline of the first job of economic batches. Since this deadline is greater than $\tilde{d}_i$ and $C_j^1 + l \leq \tilde{d}_i$, so all jobs are not late. Thirdly, in $Se(i, j)$ from job u, we swap the jobs of batch B with its previous job in economic batch, in order that batch B is before the economic batches from job u. Batch B departs at its completion time. In this action, we anticipate D_B, so all jobs are not late. And we delay the deadline of the first job of each economic batch, so all jobs are not late. If in the first step we can not find the job u, we ignore the third step. At the end, the jobs of batch B are processed consecutively on machine M_1. Figure 4 shows an example, where express batch B is split by economic batch B_1 such that $D_{B_1} - l_1 < C_j^1$, and economic batch B_2 such that $D_{B_1} - l_1 \geq C_j^1$. After a

series of swapping, we have a new schedule, such that the jobs of batch B are processed consecutively. Applying the same rule, we assemble each split express batch to obtain a new schedule $\sigma^{'}$ which has the same transportation cost as σ.

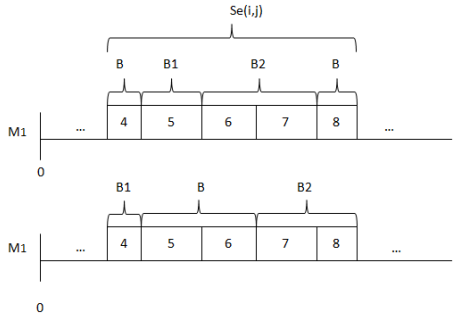


Fig. 4 Illustration of assembling split express batch for property 3

We assume batch B' is the first economic batch in schedule $\sigma^{'}$, in which the jobs are not processed consecutively on machine M_1. With the same reason as batch B, we known that batch B' is only split by express batches. We use the same definition for $Se(i,j)$. Furthermore, the express batches in $Se(i,j)$ are non-split batches. We swap the jobs of batch B' with its following job in express batch in $Se(i,j)$, in order that batch B' is after the express batches. We update the departure date for each new express batch by its completion time. In this action, we delay the deadline of the first job of batch B', so the jobs are not late. And we anticipate the deadline of the first job of express batches. Since this deadline is greater than $\tilde{d}_i$ and $C_j^1 + l \leq \tilde{d}_i$, so all jobs are not late. At the end, the jobs of batch B' are processed consecutively on machine M_1. Figure 5 shows an example, where economic batch B' is split by a non-split express batch B_1. After a series of swapping, we have a new schedule, such that the jobs of batch B' are processed consecutively. Applying the same rule, we assemble each split economic batch to obtain a new schedule which has the same transportation cost as σ. This new schedule which respects property 3 is optimal.

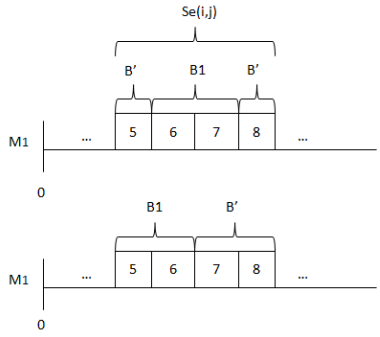


Fig. 5 Illustration of assembling split economic batch for property 3

According to property 3 of Lemma 1, the jobs of any batch are processed consecutively on machine M_1. For sections 3.2.1 and 3.2.2, we assume that the jobs are indexed in the order of completion time like $C_1^1 \leq \ldots \leq C_n^1$.

3.2.1 3PL adjusts

In many contexts, the manufacturer may not allow the 3PL to bargain (Chen and Hall [5]). We consider the 3PL's problem when he can not negotiate with the manufacturer. We give a dynamic programming algorithm for this problem. In such algorithm we use, as *state*, (i, j, s, v, m). This state corresponds to the situation in which:

- the first i jobs have been delivered,
- the currently last batch consists of jobs $j, \ldots, i$,
  - if $m = 1$ (economic transportation), the last batch departs at t_s, and at this time, v economic vehicles are used including this batch,
  - if $m = 2$ (express transportation), the last batch departs at C_i^1.

We are now in the position of describing the solution algorithm.

Algorithm 1
Value Function
$f(i, j, s, v, m)$ = the minimum transportation cost of all schedules in state (i, j, s, v, m).
Initialisation
$f(i, j, s, v, m) = \infty, \forall i = 0, \ldots, n, j = 0, \ldots, n, s = 0, \ldots, L, v = 0, \ldots, v_s, m = 1, 2$.
Boundary Condition
$f(0, 0, 0, 0, 1) = 0$, $f(0, 0, 0, 0, 2) = 0$.
Optimal Solution Value
$\min_{(j,s,v,m)\in A} f(n, j, s, v, m)$,
where $A = \{(j, s, v, m) | 1 \leq j \leq n, 0 \leq s \leq L, 0 \leq v \leq v_s, m = 1, 2\}$.
Recurrence Relation
For state (i, j, s, v, m), such that $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, i\}$, $s \in \{0, \ldots, L\}$, $v \in \{0, \ldots, v_s\}$, $m \in \{1, 2\}$, if $m = 1$, then $t_s + l_1 \leq C_j^1 + T$, $C_i^1 \leq t_s$, $s > 0$ and $v > 0$, if $m = 2$, then $C_i^1 + l_2 \leq C_j^1 + T$,
$f(i, j, s, v, m) =$

$$\min \begin{cases} f(i-1, j, s, v, m), & \text{if } i > j, m = 1, i - j + 1 \leq c_1 \\ f(i-1, j, s, v, m), & \text{if } i > j, m = 2, i - j + 1 \leq c_2 \\ \min_{(j',m',s',v')\in A_1} f(i-1, j', s', v', m') + h_1, & \text{if } i = j, m = 1 \\ \min_{(j',m')\in A_2} f(i-1, j', s, v, m') + h_2, & \text{if } i = j, m = 2 \end{cases}$$

where

$A_1 = \{(j', m', s', v') | 1 \leq j' \leq i-1, m' = 1, 2, 0 \leq s' \leq s, 0 \leq v' \leq v_{s'}, if\ s' = s, v' = v - 1\}$,

$A_2 = \{(j', m') | 1 \leq j' \leq i-1, m' = 1, 2\}$.

In the recurrence relation, the first term corresponds to the case where job i and job $i-1$ are in the same economic batch. The second term considers the case where job i and job $i-1$ are in the same express batch. In the third term, job i is in a new economic batch. This term considers all possible states of first $i-1$ jobs. In the fourth term, job i is in a new express batch.

Theorem 1 *Algorithm 1 finds an optimal distribution schedule for the 3PL's problem when the manufacturer dominates and the 3PL adjusts in polynomial time* $O((nLv_{\max})^2)$.

Proof There exists $O(n^2 L v_{\max})$ states. In the recurrence relation, the first and second terms require $O(1)$ time and are applied respectively for $O(n^2 L v_{\max})$ states. The third term requires $O(nLv_{\max})$ time and is applied for $O(nLv_{\max})$ states. The fourth term requires $O(n)$ time and is applied for $O(nLv_{\max})$ states. Therefore, the complexity of algorithm 1 is $O((nLv_{\max})^2)$. Since $v_{\max} \leq n$, this problem is solvable in polynomial time.

3.2.2 3PL negotiates

We consider the 3PL's problem when he can negotiate with the manufacturer. As discussed in the section 2, the 3PL's problem is to find a distribution schedule minimizing TC subject to makespan constraint, i.e. $C_{\max} \leq C_{\max}(\sigma)$, where $C_{\max}(\sigma) = C_{\max}(\pi^J) + T$ (see (1)). We guarantee the makespan constraint by updating deadline for each job in the following way.

$$\tilde{d}_j = C_{\max}(\sigma) - \sum_{u=j}^{n} p_j^2, \forall j = 1, \ldots, n. \tag{2}$$

We define $T_j = \tilde{d}_j - C_j^1$ for $j = 1, \ldots, n$. The 3PL aims at finding an optimal distribution schedule, minimizing TC, subject to the release time C_j^1 and the deadline $\tilde{d}_j = C_j^1 + T_j$ of each job $j, j = 1, \ldots, n$. Comparing the case where the 3PL adjusts, we note that the only difference is that T is replaced by T_j in the recurrence relation of algorithm 1. So this modified algorithm solves this 3PL's problem. Then the 3PL proposes the new production schedule to the manufacturer, in which the sequence of production on two machines is the same as schedule σ, and only the starting time of jobs on M_2 may have been changed. Since this new production schedule does not increase the makespan, it is easy for the 3PL to persuade the manufacturer to accept it.

4 Manufacturer and 3PL cooperate

In this section, firstly we give a Mixed Integer Program for integrated scheduling problem that allows generating efficient solutions that cannot be improved on one criterion without being depreciated on the other one. These solutions can be used as input for negotiation between the manufacturer and the 3PL. Secondly we provide the NP-completeness for this problem. Finally we give algorithms for some special cases which are polynomially solvable.

4.1 Mixed Integer Program

As discussed in the section 2, we consider the permutation problem, i.e. the production schedule has the same sequence on two machines. For a given production schedule, i.e. we know the sequence and starting time of each job on two machine, the distribution scheduling problem is to find an optimal distribution schedule, minimizing TC, subject to the release time C_j^1 and the deadline $\tilde{d}_j$ of each job $j, j = 1, \ldots, n$. This is the 3PL's problem when the manufacturer dominates and the 3PL adjusts. So the properties of Lemma 1 hold for the distribution schedule in this case, i.e. the manufacturer and the

3PL cooperate. A solution of this problem is completely characterized by a sequence of batches, which is associated to a transportation mode, a delivery departure date, and a starting time on two machines. The order of jobs in each batch is irrelevant. We give a MIP (Mixed Integer Program) for integrated scheduling problem.

We introduce the following notation.

$$u_{cs} = \begin{cases} 1, \text{ if the economic vehicle } c \text{ is available at the time } t_s, c = 1, \ldots, V_1, \\ \quad s = 1, \ldots, L \\ 0, \text{ otherwise} \end{cases}$$

Decision variables

- t_k^e : starting time of batch k on machine M_e, $k = 1, \ldots, n, e = 1, 2$
- $x_{ik} = \begin{cases} 1, \text{ if job } i \text{ is in batch } k, i = 1, \ldots, n, k = 1, \ldots, n \\ 0, \text{ otherwise} \end{cases}$
- $w_k^m = \begin{cases} 1, \text{ if batch } k \text{ contains at least one job and is delivered by transport mode } m, \\ \quad k = 1, \ldots, n, m = 1, 2 \\ 0, \text{ otherwise} \end{cases}$
- $y_{kc} = \begin{cases} 1, \text{ if batch } k \text{ contains at least one job and is delivered by economic vehicle } c, \\ \quad k = 1, \ldots, n, c = 1, \ldots, V_1 \\ 0, \text{ otherwise} \end{cases}$

$$\min C_{\max} \tag{3}$$

$$\min \sum_{k=1}^{n} w_k^1 h_1 + \sum_{k=1}^{n} w_k^2 h_2 \tag{4}$$

$$\text{subject to } \sum_{k=1}^{n} x_{ik} = 1, i = 1, \ldots, n \tag{5}$$

$$\sum_{m=1}^{2} w_k^m \leq 1, k = 1, \ldots, n \tag{6}$$

$$\sum_{i=1}^{n} x_{ik} \leq \sum_{m=1}^{2} c_m w_k^m, k = 1, \ldots, n \tag{7}$$

$$t_{k+1}^1 - t_k^1 \geq \sum_{i=1}^{n} p_i^1 x_{ik}, k = 1, \ldots, n-1 \tag{8}$$

$$t_{k+1}^2 - t_k^2 \geq \sum_{i=1}^{n} p_i^2 x_{ik}, k = 1, \ldots, n-1 \tag{9}$$

$$t_k^2 - t_k^1 \geq \sum_{i=1}^{n} p_i^1 x_{ik} + l w_k^2, k = 1, \ldots, n \tag{10}$$

$$\sum_{c=1}^{V} y_{kc} = w_k^1, k = 1, \ldots, n \tag{11}$$

$$t_k^1 + \sum_{i=1}^{n} p_i^1 x_{ik} \leq \sum_{s=1}^{L} \sum_{c=1}^{V} t_s u_{cs} y_{kc} + w_k^2 \sum_{i=1}^{n} p_i^1, k = 1, \ldots, n \tag{12}$$

$$t_k^2 \geq \sum_{s=1}^{L} \sum_{c=1}^{V} t_s u_{cs} y_{kc} + l w_k^1, k = 1, \ldots, n \tag{13}$$

$$C_{\max} \geq t_n^2 + \sum_{i=1}^{n} p_i^2 x_{in} \tag{14}$$

$$t_k^1 \geq 0, k = 1, \ldots, n \tag{15}$$

$$t_k^2 \geq 0, k = 1, \ldots, n \tag{16}$$

$$x_{ik} \in \{0, 1\}, i = 1, \ldots, n, k = 1, \ldots, n \tag{17}$$

$$w_k^m \in \{0, 1\}, k = 1, \ldots, n, m = 1, 2 \tag{18}$$

$$y_{kc} \in \{0, 1\}, k = 1, \ldots, n, c = 1, \ldots, V \tag{19}$$

In the MIP, (3) and (4) are the objective functions. Constraints (5) ensure that each job is assigned to one batch. Constraints (6) ensure that one batch either is delivered by one type of vehicle or does not have any job. Constraints (7) represent the constraint of capacity of vehicle. Constraints (8) and (9) guarantee the assumption of permutation. Constraints (10) represent the constraints of delivery time when one batch is delivered by an express vehicle. Constraints (11) ensure that one economic vehicle is assigned to a batch if necessary. Constraints (12) mean that any batch delivered by an economic vehicle should be available at the fixed departure date for this economic vehicle. The part $w_k^2 \sum_{i=1}^{n} p_i^1$ guarantees that if the batch is delivered by one express vehicle, these constraints are always true. Constraints (13) represent the constraints of delivery time when one batch is delivered by an economic vehicle. Constraints (14) define the makespan $C_{\max}$). Constraints (15) – (19) give the domain of definition of each variable.

4.2 Complexity

Esswein et al. [8] prove that the integrated scheduling problem, in the case where there is only express mode with unlimited number of vehicles and unbounded capacity is NP-hard. For bounded capacity, Aloulou et al. [2] prove that this problem is NP-hard when $c_2 \geq 3$ even if $p_j^1 = p_j^2 = p_j$ for $j = 1, \ldots, n$. However the problem with $c_2 = 2$ is open. Therefore, in the following, we consider the problem where there is only economic transportation mode with limited number of vehicles.

Theorem 2 *The integrated scheduling problem is NP-hard, even if* $p_j^1 = p_j^2 = p_j$ *for* $j = 1, \ldots, n$, $V_2 = 0$ *and* $l = 0$

Proof We use the following NP-complete problem: 3-partition. Given $3n$ integers $a_1, \ldots, a_{3n}$, so that $\sum a_j = nW$, and such that $W/4 < a_j < W/2$ for all j, is it possible to partition them into n triples each summing up to W?

Given an instance of 3-partition, define an instance of integrated problem as follows. There are $4n$ jobs, namely n long jobs and $3n$ short jobs. The short jobs correspond to the integers of 3-partition, so $p_j^1 = p_j^2 = p_i = a_j$ for them. The long jobs have the same processing time $p_j^1 = p_j^2 = p_j = Q >> W$. There is only economic transportation mode and transportation time l equals to zero. There are n periodic fixed departure dates $Q + W, 2(Q + W), \ldots, n(Q + W)$. We distinguish different cases.

– case where $c_1 \geq 4$. For each fixed departure date, there is one economic vehicle. We want to know whether there exists a schedule σ consisting of n batches, such that $C_{\max} \leq (n + 1)(W + Q)$ et $TC \leq nh_1$.

– case where $c_1 = 2$ or 3. For each fixed departure date, there are two economic vehicles. We want to know whether there exists a schedule σ consisting of $2n$ batches, such that $C_{\max} \leq (n+1)(W+Q)$ et $TC \leq 2nh_1$.
– case where $c_1 = 1$. For each fixed departure date, there are four economic vehicles. We want to know whether there exists a schedule σ consisting of $4n$ batches, such that $C_{\max} \leq (n+1)(W+Q)$ et $TC \leq 4nh_1$.

Suppose that a solution Σ to 3-partition exists. Then we can build a schedule σ for integrated problem in the three different cases as follows.

– case where $c_1 \geq 4$. We put in each batch one long job and one of the triples of Σ. Doing so, the total processing time of each batch is exactly $Q+W$, and each batch departs at its completion time which is exactly one fixed departure date. We have $C_{\max} = (n+1)(W+Q)$ et $TC = nh_1$.
– case where $c_1 = 2$ or 3. We apply the same rule as the case where $c_1 \geq 4$ and replace one batch by two batches. We have $C_{\max} = (n+1)(W+Q)$ et $TC = 2nh_1$.
– case where $c_1 = 1$. We apply the same rule as the case where $c_1 \geq 4$ and replace one batch by four batches. We have $C_{\max} = (n+1)(W+Q)$ et $TC = 4nh_1$.

So the obtained schedule σ is a solution to integrated problem.

Now we suppose that a solution σ to integrated problem exists. The following facts concerning σ hold.

Fact 1. There does not exist idle time between jobs on both machines. If there exists idle time on M_1, there would have at least one job exceed the last fixed departure date $n(Q+W)$, and we do not have any feasible solution in this case. We must have idle time at the beginning of M_2, which has length $Q+W$. Since the total processing time of jobs on M_2 is $n(Q+W)$, so there does not exist idle time between jobs on M_2.

Fact 2. At each fixed departure date, jobs with total processing time which equals to $Q+W$ are delivered. If not, there would have idle time on M_2. So at each fixed departure date, the batch(es) is(are) formed by one long job and three short jobs, i.e. the total processing time of the short jobs equals to W.

In conclusion, the short jobs delivered at each fixed departure date define a triple which constitutes a solution to 3-partition.

4.3 Special cases

We consider a special case where all jobs have a same processing time on M_1, i.e. $p_j^1 = p^1$, for $j = 1, \ldots, n$. The following property holds for this special case.

Lemma 2 *For any non-dominated point, there exists an efficient solution for integrated scheduling problem in the case, where $p_j^1 = p^1$, for $i = 1, \ldots, n$, such that jobs are sequenced in the order of non-increasing processing time on machine M_2.*

Proof Suppose there exists an efficient solution σ, such that job i is processed after job j on M_2 and $p_i^2 > p_j^2$. We can swap two jobs in integrated schedule, i.e. we swap job i and job j on both machines and on their delivery batches. In this action, we do not change the transportation cost and do not increase the processing time between job i and job j on M_2. Applying the same rule, we can swap the jobs to obtain a new schedule which respects the property of Lemma 2 and has the same $C_{\max}$ and TC as schedule σ. So this new schedule is an efficient solution associated to the same non-dominated point $(C_{\max}(\sigma), TC(\sigma))$.

According to Lemma 1 and Lemma 2, we use algorithm 2 to solve this special case in polynomial time.

Algorithm 2

1. Index jobs in the order of non-increasing processing time on M_2, such that $p_1^2 \geq \ldots \geq p_n^2$. The sequences on both machine follow the order of increasing index of jobs.
2. Find all efficient solutions by the following dynamic programming algorithm.
   Value Function
   $f(i, j, s, v, m, n_1, n_2)$ = the minimum $C_{\max}$ of all schedules in state $(i, j, s, v, m, n_1, n_2)$. In state $(i, j, s, v, m, n_1, n_2)$, we have delivered the first i jobs.The last batch consists of job $j, \ldots, i$. m represents the transportation mode of the last batch, such that $m = 1$ represents the economic transportation, while $m = 2$ represents the express transportation. If $m = 1$, the last batch departs at t_s, and at this time, v economic vehicles are used including this batch. If $m = 2$, the last batch departs at C_i^1. n_1 and n_2 are the number of delivery batches of economic transportation and express transportation, respectively.
   Initialisation
   $f(i, j, s, v, m, n_1, n_2) = \infty, \forall i = 0, \ldots, n, j = 0, \ldots, n, s = 0, \ldots, L, v = 0, \ldots, v_s, m = 1, 2, n_1 = 0, \ldots, n, n_2 = 0, \ldots, n$.
   Boundary Condition
   $f(0, 0, 0, 0, 1, 0, 0) = 0$, $f(0, 0, 0, 0, 2, 0, 0) = 0$.
   Optimal Solution Value
   For each $n_1 = 0, \ldots, n, n_2 = 0, \ldots, n$, we search $\min_{(j,s,v,m)\in A} f(n, j, s, v, m, n_1, n_2)$ which is not infinity, and $A = \{(j, s, v, m) | 1 \leq j \leq n, 0 \leq s \leq L, 0 \leq v \leq v_s, m = 1, 2\}$.
   Recurrence Relation
   For state (i, j, s, v, m), such that $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, i\}$, $s \in \{0, \ldots, L\}$, $v \in \{0, \ldots, v_s\}$, $m \in \{1, 2\}$, $n_1 =\in \{0, \ldots, n\}$, $n_2 =\in \{v, \ldots, n\}$, if $m = 1$, then $C_i^1 \leq t_s$, $s > 0$ and $v > 0$, if $m = 2$, then $n_2 > 0$,
   $f(i, j, s, v, m, n_1, n_2) =$

$$\min \begin{cases} f(i-1, j, s, v, m, n_1, n_2) + p_i^2, & \text{if } i > j, m = 1, i - j + 1 \leq c_1 \\ \max\{f(i-1, j, s, v, m, n_1, n_2) - \sum_{u=j}^{i-1} p_u^2, & \\ C_u^1 + l\} + \sum_{u=j}^{i} p_u^2, & \text{if } i > j, m = 2, i - j + 1 \leq c_2 \\ \max\{\min_{(j',m',s',v')\in A_1} f(i-1, j', s', v', m', & \\ n_1 - 1, n_2), t_s + l\} + p_j^2, & \text{if } i = j, m = 1 \\ \max\{\min_{(j',m')\in A_2} f(i-1, j', s, v, m', n_1, & \\ n_2 - 1), C_u^1 + l\} + p_j^2, & \text{if } i = j, m = 2 \end{cases}$$

   where
   $A_1 = \{(j', m', s', v') | 1 \leq j' \leq i - 1, m' = 1, 2, 0 \leq s' \leq s, 0 \leq v' \leq v_{s'}, if \ \ s' = s,$ then $v' = v - 1\}$,
   $A_2 = \{(j', m') | 1 \leq j' \leq i - 1, m' = 1, 2\}$.

In the recurrence relation, the first term corresponds to the case where job i and job $i-1$ are in the same economic batch. The first term corresponds to the case where job i and job $i - 1$ are in the same express batch. In the third term, job i is in a new economic batch. The function max is to find the earliest starting time on M_2 for job

j. In the fourth term, job i is in a new express batch. The function max is to find the earliest starting time on M_2 for job j.

The step 1 of algorithm 2 requires $O(n \log n)$ time. In the dynamic programming algorithm, there exists $O(n^4 L v_{\max})$ states. In the recurrence relation, the first term and second term require $O(1)$ time and are applied for $O(n^4 L v_{\max})$ states respectively. The third term requires $O(nLv_{\max})$ time and is applied for $O(n^3 L v_{\max})$ states. The fourth term requires $O(n)$ time and is applied for $O(n^3 L v_{\max})$ states. Therefore, the complexity of algorithm 2 is $O(n^4 (L v_{\max})^2)$. Hence we have the following result.

Theorem 3 *Algorithm 2 solves the integrated scheduling problem in polynomial time* $O(n^4 (L v_{\max})^2)$*, when* $p_j^1 = p^1$*, for* $j = 1, \ldots, n$.

Now we consider another special case where all jobs have a same processing time on M_2, i.e. $p_j^2 = p^2$, for $j = 1, \ldots, n$. With the similar proof of Lemma 2, we have that there exists an efficient solution for integrated scheduling problem in this case, such that jobs are sequenced in the order of non-decreasing processing time on M_1. Then we can firstly index jobs in the order of non-decreasing processing time on M_1, such that $p_1^1 \leq \ldots \leq p_n^1$, secondly use the dynamic program of Algorithm 2 to solve the problem in this case. We call this new algorithm as Algorithm 3. We have the following result.

Theorem 4 *Algorithm 3 solves the integrated scheduling problem in polynomial time* $O(n^4 (L v_{\max})^2)$*, when* $p_j^2 = p^2$*, for* $j = 1, \ldots, n$.

5 Benefit of cooperation

We evaluate the benefit of cooperation using numerical experiments. Denote by σ and δ the manufacturer's optimal schedule and the 3PL's optimal schedule respectively in the case where the manufacturer dominates and the 3PL adjusts. Denote by η and θ the manufacturer's optimal schedule and the 3PL's optimal schedule respectively in the case where the manufacturer dominates and the 3PL negotiates. Denote by λ an efficient integrated schedule in the case where the manufacturer and the 3PL cooperate and have the same bargaining power. For these experiments, we choose λ among the set of efficient integrated schedules such that the 3PL is favored, i.e. λ minimizes TC subject to the following constraint, $C_{\max}(\lambda) \leq C_{\max}(\sigma)$. Firstly, we evaluate the benefit of cooperation in the case where the manufacturer dominates. The 3PL's benefit in this case is defined by $(TC(\delta) - TC(\theta))/TC(\delta)$. The manufacturer's benefit in this case is defined by $(C_{\max}(\sigma) - C_{\max}(\eta))/C_{\max}(\sigma)$. Secondly, we evaluate the benefit of cooperation in the case where the manufacturer and the 3PL have the same bargaining power and cooperate. The 3PL's benefit in this case is defined by $(TC(\delta) - TC(\lambda))/TC(\delta)$. The manufacturer's benefit in this case is defined by $(C_{\max}(\sigma) - C_{\max}(\lambda))/C_{\max}(\sigma)$.

We generate the parameters as follows. In these first experiments, we consider small values of the number of jobs n, i.e. $n \in \{5, 10, 20\}$. p_i^1 and p_i^2 are randomly generated integers from $[1, 100]$. We consider three values for the capacity c_1, i.e. $c_1 \in \{2, 3, 5\}$. We use parameter α to give the relationship between c_1 and c_2, such that $c_2 = \lfloor \alpha c_1 \rfloor$, $\alpha \in \{0.5, 1\}$. We have $h_1 = 1$ and use parameter β to give the relationship between h_1 and h_2, such that $h_2 = \beta \frac{c_2}{c_1} h_1$, $\beta \in \{1, 2\}$. The transportation time l is a randomly generated integer from $[1, 100]$. The number of economic vehicles $V_1 = \lceil \epsilon \frac{n}{c_1} \rceil$, where ϵ is randomly generated in interval $[0.6, 1.4]$. Denote by $\overline{v}$ the average number of economic vehicles for each fixed departure date and we consider three values for $\overline{v}$, i.e. $\overline{v} \in$

$\{1, 2, 3\}$. We determine the value of L like $L = \lceil \frac{V_1}{\overline{v}} \rceil$. We assign $\overline{v} - 1$ economic vehicles for fixed departure date t_s where $s = 1, \ldots, \overline{v}L - V_1$, and $\overline{v}$ for each remaining fixed departure date. We assign L fixed departure dates periodically and periodicity t is determined as $t = \lceil \frac{\sum_{i=1}^{n} p_i^1}{L} \rceil$. We determine the value of T like $T = \lceil \gamma(t + l) \rceil$, where γ is randomly generated from $[0.6, 1.4]$. We generate 108 series of instances with the following parameters, n, c_1, α, $\overline{v}$ and β. We generate 10 instances for each series of instances. So we generate 1080 instances in total. We evaluate the benefit of cooperation for different values of the four parameters, n, α, β and $\overline{v}$. (see table 3 and table 4) The algorithms were implemented in C++ and were run on a Intel(R) Core(TM) i5-2520M CPU @ 2.50GHz processor. The MIP was solved using CPLEX 12.2. The table 2 give the average computational time of algorithm 1 and of MIP. Since the MIP can not solve the instances when $n = 50$, we evaluate the benefit of cooperation for the instances when $n \in \{5, 10, 20\}$.

n	**time for algorithm 1 seconds**	**time for MIP seconds**
5	0.02	0.26
10	0.3	0.92
20	1.34	27.56

Table 2 Average computational time of algorithm 1 and of MIP

		Manufacturer dominates	
		Manufacturer's benefit	**3PL's benefit**
Overall averages		12.41%	7.44%
n	5	13.49%	3.87%
	10	12.96%	7.51%
	20	10.78%	10.95%
α	0.5	13.29%	6.16%
	1	11.98%	8.55%
β	1	16.29%	1.95%
	2	8.98%	12.77%
$\overline{v}$	1	9.47%	12.13%
	2	13.31%	5.84%
	3	15.13%	4.1%

Table 3 Benefits of cooperation when the manufacturer dominates

Our results show significant benefit of cooperation for both actors. The manufacturer's benefit is 12.41% and the 3PL's benefit is 7.44% in the case where the manufacturer dominates, the manufacturer's benefit is 16.09% and the 3PL's benefit is 8.41% in the case where the manufacturer and the 3PL cooperate. Along with the increasing of value of n, the manufacturer's benefit decreases and the 3PL's benefit increases. The results of parameter α show that the capacity of vehicle has a weak effect on the benefit. $\alpha = 0.5$ means $c_2 = \lfloor 0.5c_1 \rfloor$, and $\alpha = 1$ means $c_1 = c_2$. The results of parameter β show that the transportation cost per delivery has a strong effect on the benefit. $\beta = 1$ means $h_2/c_2 = h_1/c_1$, and $\beta = 2$ means $h_2/c_2 = 2h_1/c_1$. When the transportation cost of express vehicle is twice as expensive as that of economic vehicle, the 3PL's benefit is much more than that when two types of vehicle have the same

		Manufacturer and 3PL cooperate	
		Manufacturer's benefit	**3PL's benefit**
Overall averages		16.09%	8.41%
n	5	18.85%	4.42%
	10	17.26%	9.01%
	20	12.16%	11.81%
α	0.5	17.58%	7.16%
	1	15.41%	9.62%
β	1	22.08%	1.96%
	2	10.91%	14.82%
$\overline{v}$	1	11.63%	13.62%
	2	17.38%	6.85%
	3	20.47%	4.69%

Table 4 Benefits of cooperation when the manufacturer and the 3PL have the same bargaining power and cooperate

transportation cost, i.e. 12.77% vs 1.95% and 14.82% vs 1.96%. The manufacture has a contrary situation. The results of parameter $\overline{v}$ show that the dispersion of fixed departure dates has a strong effect on the benefit. Since $L = \lceil \frac{V_1}{\overline{v}} \rceil$ and $t = \lceil \frac{\sum_{i=1}^{n} p_i^1}{L} \rceil$, with the same number of economic vehicles V_1, $\overline{v} = 1$ represents the smallest periodicity t of fixed departure date and $\overline{v} = 3$ represents the longest periodicity. When the periodicity t increases, the 3PL's benefit decreases significantly and the manufacturer's benefit increases significantly.

6 Conclusion

In this paper, we consider scheduling problem in a supply chain with two actors, a manufacturer and a third-party logistics(3PL), and interplant distribution between the upstream stage and the downstream stage of production. The manufacturer's objective is to minimize makespan $C_{\max}$ and the 3PL's objective is to minimize transportation cost TC. we investigate several scheduling problems in three types of contract: the manufacturer dominates, the 3PL adjusts; the manufacturer dominates, the 3PL negotiates; the manufacturer and the 3PL cooperate and have the same bargaining power. We give an algorithm to solve the 3PL's problem in the case with the first type of contract, where there is not cooperation between two actors. In the case with the second type of contract, we give an algorithm to solve the 3PL's problem when he can negotiate with the manufacturer. In the case with the third type of contract, the two actors can cooperate to determine an integrated schedule. We prove the NP-completeness for the 3PL's problem and give the algorithms for the special cases, i.e. all jobs have a same processing time on M_1 or M_2. In our first experiments, we evaluate the benefit of cooperation in the second type and the third type of contract. Our first results show significant benefit of cooperation for both actors in these two cases.

Several important research issues remain open for future investigation. A first important research is to develop a heuristic algorithm for the integrated scheduling problem. Another important issue is to investigate a cooperation mechanism, which guide the two actors to cooperate. Finally, we extend our model to consider a three stages supply chain with interplant and outbound distribution, where a manufacturer has to process a set of jobs at the upstream stage and the downstream stage. A 3PL is

in charge of transportation of semi-finished products from the upstream stage to the downstream stage, and then finished products to clients.

Acknowledgements This work was partially sponsored by the French National Research Agency: project ANR-09-BLAN-0361-02 (GUEPARD).

References

1. Agnetis, A., Hall, N.G., Pacciarelli, D.: Supply chain scheduling: Sequence coordination. Discrete Applied Mathematics **154**(15), 2044–2063 (2006)
2. Aloulou, M.A., Bouzaiene, A., Dridi, N., Vanderpooten, D.: A bicriteria two-machine flow-shop serial-batching scheduling problem with bounded batch size. Journal of Scheduling (2012). Submitted, under second round review
3. Aydinliyim, T., Vairaktarakis, G.L.: Coordination of outsourced operations to minimize weighted flow time and capacity booking costs. Manufacturing & Service Operations Management **12**(2), 236–255 (2010)
4. Chen, Z.L.: Integrated production and outbound distribution scheduling: Review and extensions. Operations Research **58**(1), 130–148 (2010)
5. Chen, Z.L., Hall, N.G.: Supply chain scheduling: Conflict and cooperation in assembly systems. Operations Research **55**(6), 1072–1089 (2007)
6. Dawande, M., Geismar, H.N., Hall, N.G., Sriskandarajah, C.: Supply chain scheduling: Distribution systems. Production and Operations Management **15**(2), 243–261 (2006)
7. Erenguc, S.S., Simpson, N.C., Vakharia, A.J.: Integrated production/distribution planning in supply chains: An invited review. European Journal of Operational Research **115**(2), 219–236 (1999)
8. Esswein, C., Billaut, J.C., Strusevich, V.A.: Two-machine shop scheduling: Compromise between flexibility and makespan value. European Journal of Operational Research **167**(3), 796–809 (2005)
9. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: E.J. P.L. Hammer, B. Korte (eds.) Discrete Optimization II, *Annals of Discrete Mathematics*, vol. 5, pp. 287–326. Elsevier (1979)
10. Hall, N.G., Potts, C.N.: Supply chain scheduling: Batching and delivery. Operations Research **51**(4), 566–584 (2003)
11. Johnson, S.M.: Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly **1**(1), 61–68 (1954)
12. Lee, C.Y., Chen, Z.L.: Machine scheduling with transportation considerations. Journal of Scheduling **4**(1), 3–24 (2001)
13. Lee, C.Y., Strusevich, V.A.: Two-machine shop scheduling with an uncapacitated interstage transporter. IIE Transactions **37**(8), 725–736 (2005)
14. Li, K., Sivakumar, A.I., Ganesan, V.K.: Analysis and algorithms for coordinated scheduling of parallel machine manufacturing and 3pl transportation. International Journal of Production Economics **115**(2), 482–491 (2008)
15. Sarmiento, A.M., Nagiy, R.: A review of integrated analysis of production-distribution systems. IIE Transactions **31**(11), 1061–1074 (1999)
16. Thomas, D.J., Griffin, P.M.: Coordinated supply chain management. European Journal of Operational Research **94**, 1–15 (1996)
17. Zhong, W., Chen, Z.L., Chen, M.: Integrated production and distribution scheduling with committed delivery dates. Operations Research Letters **38**(2), 133–138 (2010)

MISTA 2013

Automatic timetabling in Higher Education Institutions: a real scenario and solution

Pedro Fernandes • Carla Sofia Pereira • Armando Barbosa

Abstract In a time where the need to reduce costs has become part of the day to day reality of all Educational Institutions, it is unthinkable to continue manually performing those tasks that can be automated and optimized – the creation of timetables. The automatic creation of timetables for Education is one of the most studied problems by the scientific community. However, almost all studies have been based on very simplified models of reality, not having thus a practical application. A realistic model of the problem, robust algorithms that are able to find valid solutions in highly restricted environments and optimization methods that are able to quickly provide quality results, are key factors to consider when attempting to solve the (real) problem faced by Educational Institutions. This paper presents a summary of the work performed by Bullet Solutions over the last few years, from the first stage of understanding and modelling the problem, until the final analysis of the results obtained with the use of the developed software under real conditions.

1 Introduction

The problem of creating timetables for Educational Institutions is typically defined as the scheduling of a set of lessons involving teachers and students, on a set of classrooms, in certain time slots, considering a number of constraints [1], [2] and [3].

Throughout an academic year, in every Educational Institution, a considerable number of days and human resources are spent trying to find, manually, a solution that respects all the existing rules (a feasible solution) and that, at the same time, can meet the expectations of all participants (a quality solution).

The differences between the academic systems in each country and even between institutions in the same country, originates a significant amount of models and formulations of

Pedro Fernandes
Bullet Solutions, Porto, Portugal
E-mail: pedro.fernandes@bulletsolutions.com

Carla Sofia Pereira
CIICESI-ESTGF, Instituto Politécnico do Porto, Felgueiras, Portugal
E-mail: carla.pereira@eu.ipp.pt

Armando Barbosa
Bullet Solutions, Porto, Portugal
E-mail: armando.barbosa@bulletsolutions.com

the problem, varying the structure and the dependence on the base information, the set of used constraints or the quality concept of the solution [1], [2] and [4].

Due to its combinatorial nature and associated complexity, this is one of the most studied problems by the scientific community and by the Operational Research area in particular [1], [3] and [5]. However, almost all studies have been based on very simplified models of reality, not having thereby a practical application.

In a time where the need to reduce costs has become part of the day to day reality of all Educational Institutions, it is unthinkable to continue manually performing those tasks that can be automated and optimized. It is possible to produce solutions faster and with better quality, easing the future management of changes throughout the year, and relieving the responsible team to perform other tasks that really add value.

This paper presents some of the main results of the work carried out by Bullet Solutions, which started in 2005 with a detailed identification and analysis of requirements, with the goal of building a real and comprehensive model of the problem. This model was used by the end of 2006 as the basis for the first version of the product Bullet TimeTabler Education (BTTE), an automatic and optimized generator of timetables, which has been updated and improved over the years with the contributions from almost all the Portuguese Higher Education Schools (Universities and Polytechnic Institutes), as well as some foreign institutions.

Unlike the claims of some studies, it was possible to build a model and thereafter a complete product that is adapted to different specific needs. At present, the BTTE software is successfully used in more than half of the Portuguese Higher Education Schools, including the 10 major ones.

This paper is organized into seven main sections. In order to compare with related work concerning the production of timetables for Educational Institutions, section 2 presents a review of the state of the art. Section 3 describes the approach used to define the model of the problem, which is detailed in section 4. Section 5 describes the algorithms developed for solving the problem. Some case studies and the main results are analyzed in section 6. Finally, section 7 summarizes the main conclusions of the work.

2 Related work

The automatic creation of timetables for Educational Institutions is a problem studied by the scientific community since the decade of 1960. Over almost 50 years, hundreds of studies were published, with many different formulations of the problem and solving techniques. A summary of the most relevant studies until the beginning of this century can be found in [1].

Despite the enormous attention given to this problem, the efforts put in the attempt to solve it, and the significant progress achieved, a huge gap between the models used in research and the reality found in the Educational Institutions still exists. The long way that it is still necessary to walk to bridge this gap is discussed in [6], without any kind of preconception; it provides an excellent critical analysis of the options taken in the studies conducted and the impossibility of applying them in practice. The author understands the need to restrict the problem in order to have a model for comparing studies and solving methods but, at the same time, he demonstrates that only working with real or near real models it is possible to successfully apply in practice the results of all this research.

A new study [2] published in 2012 aimed at filling the enormous gap identified in [6]. In this paper, the authors seek to create one model closer to the reality found in Educational Institutions, in order to provide a basis for further studies, looking for an approach that allows applying the research results to the real existing problems. Although an important step forward was taken, the modelling presented is still far from complete.

Even before this formal attempt to create a more comprehensive model, some approaches with interesting formulations of the problem were proposed in [7] and [8]; they already include in the analysis a large set of hard and soft constraints and goals that are forgotten in the overwhelming majority of the existing studies and that are crucial for a real model.

Three very practical approaches, with visions very focused on the final user, can be found in [4], [5] and [12]. These studies are primarily concerned with modelling real cases from Educational Institutions, looking for the most adequate solving methods at a later stage.

As mentioned, a full summary of studies carried out until the beginning of this century, with different solving techniques and results, is given in [1]. It is also worth citing one paper that compares more recent studies that handle the problem with the use of meta-heuristics [3], due to the importance and contribution of these techniques to the area in question, and another one that stresses the importance of meta-heuristics in complex combinatorial problems [9]. Also relevant are two studies ([10] and [11]) that discuss the importance and impact of a correct definition of the neighbourhood structures in solving large-scale combinatorial problems.

Despite the fact that the present work is on a different variant of the problem, it is also pertinent referring to the effort that is being made in the pursuit of standard formulations for the High School Timetabling problem [13]. Most of the components of the model that is presented in this work are clearly relevant to the University problem as well.

Besides the problem associated with the lack of a representative reference model, the gap between research and the actual needs mentioned in [6] raises a second problem. Almost all known studies focus on the optimization of the solutions found, while in many cases the greatest challenge faced by the institutions is to find a feasible solution to the problem. The fact that almost all studies only take into account a very small part of the real constraints does not put in evidence this difficulty; in such cases the quality of the final solution found is the differentiating factor of the research work. It is without doubt essential to achieve a final solution with the highest quality possible, but the first step should not be forgotten, that is, finding the adequate techniques that produce an initial feasible solution in highly constrained environments such as the overwhelming majority of Educational Institutions.

3 Main principles of the proposed approach

The best approach to understand and solve a problem is to observe it in practice, identify and try to include all the variables involved, in order to have a clear perception of it. It is not worth finding an excellent solution to a problem that does not actually exist or that was misunderstood in all its dimensions. Thus, only the identification and analysis of requirements in the field allows obtaining a clear and comprehensive vision of what currently exists and what is intended in the future.

The identification of requirements was carried out in three steps. A first phase of meetings took place between Bullet Solutions and Educational Institutions; these served to understand in a general way the type of operation, the major daily difficulties, the state of current development and future expectations. A second phase was more focused on solving the problem itself; it was possible to understand all existing flows and all variables involved, by learning all specific and particular issues with the help of those responsible for the creation of timetables. The third phase consisted in the validation of all information collected.

The number of meetings on each institution, in each of the phases, was quite variable and depended mainly on the degree of complexity of each case. About two dozens of institutions were studied, all over the country, both public and private and with quite different sizes. The main objective was to collect a set of case studies, as much comprehensive and diverse as possible, so that a complete analysis could be carried out.

Later on, with the commercialization of the first version of BTTE by the end of 2006 and the growing number of users since then, the number of case studies and contributions to enrich the model has significantly increased. This allowed a constant evolution of the line of products developed by Bullet Solutions.

Simultaneously with the identification of requirements, a review of the state of the art was carried out; it focused initially on the variety of the existing formulations of the problem in order to complement and validate all information collected in the field.

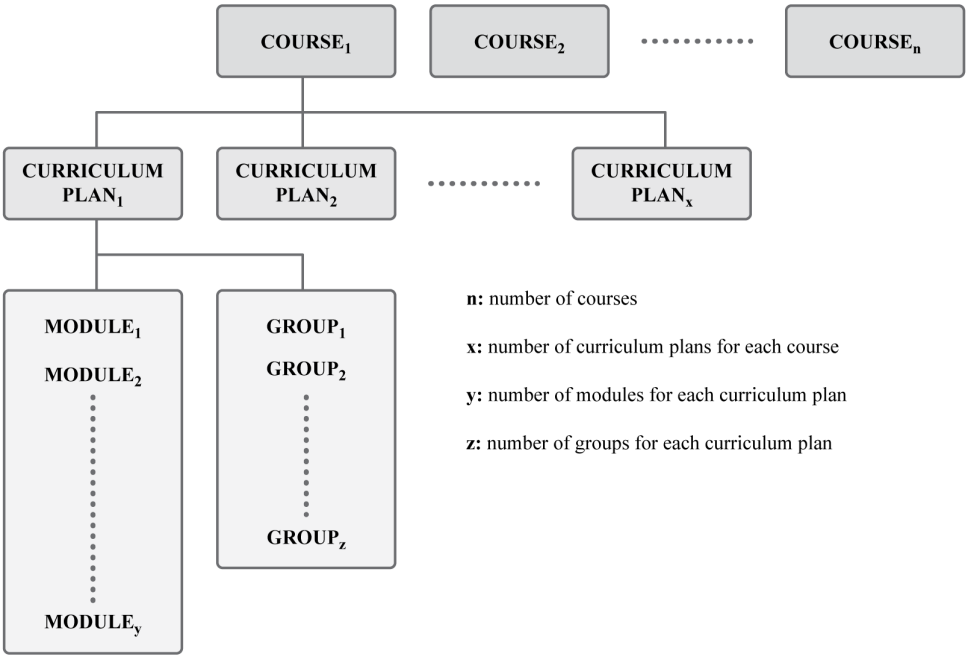
4 BTTE: Model definition

4.1 Fundamental concepts

The Portuguese educational system presents a great diversity of scenarios and complexity levels. However, these factors are much more related to specific aspects (rules that have to be fulfilled or quality objectives that must be achieved) than properly to the structure of the problem itself.

An Educational Institution may offer various **courses**. Each course can have multiple **curriculum plans**, each of which is composed of a set of **modules**. Associated with a curriculum plan there is a set of **groups**; a group is a set of students who follow this plan and share the same timetable. Figure 1 presents a schematic representation of this structure.

Figure 1 Basic concepts of a curricular structure



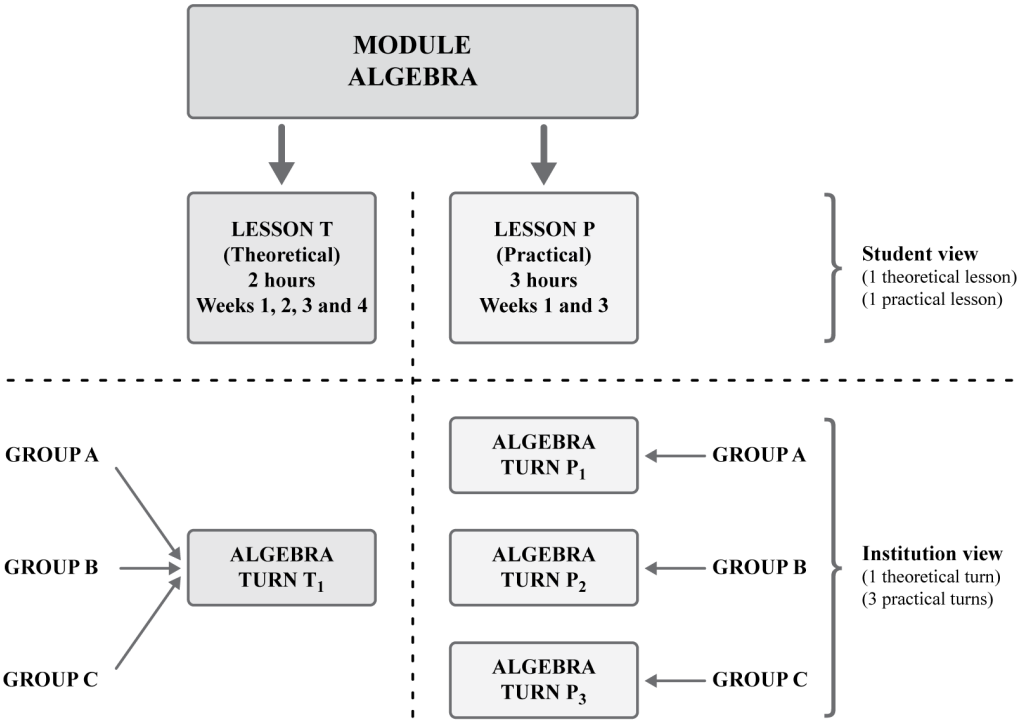
BTTE does not schedule individual students, only groups. The assignment of students to groups occurs in a separate process, where the students apply for the existing timetables, choosing the most suitable ones. In some cases, the Institution itself defines the association of students to the groups. In either case, the assignment of students to groups is not handled by BTTE because, in the Portuguese reality, it is an independent process that does not influence directly the creation of timetables. The conflicts are based on the curricula structure and the Portuguese situation can be included in the Curriculum-Based Course Timetabling problem [2].

Each institution has a range of **classrooms**, which may be located in different buildings and even in different geographical areas. Each classroom has its own capacity and a set of characteristics that differentiate it from the others. The institution also has a set of **teachers**, who teach the modules.

The **lessons** represent the way a particular module is taught, from the point of view of the student. A teaching load of one or more **types** (theoretical, practical, for example) and the **temporal space** in which it occurs (weeks) characterizes each lesson. Each lesson can have

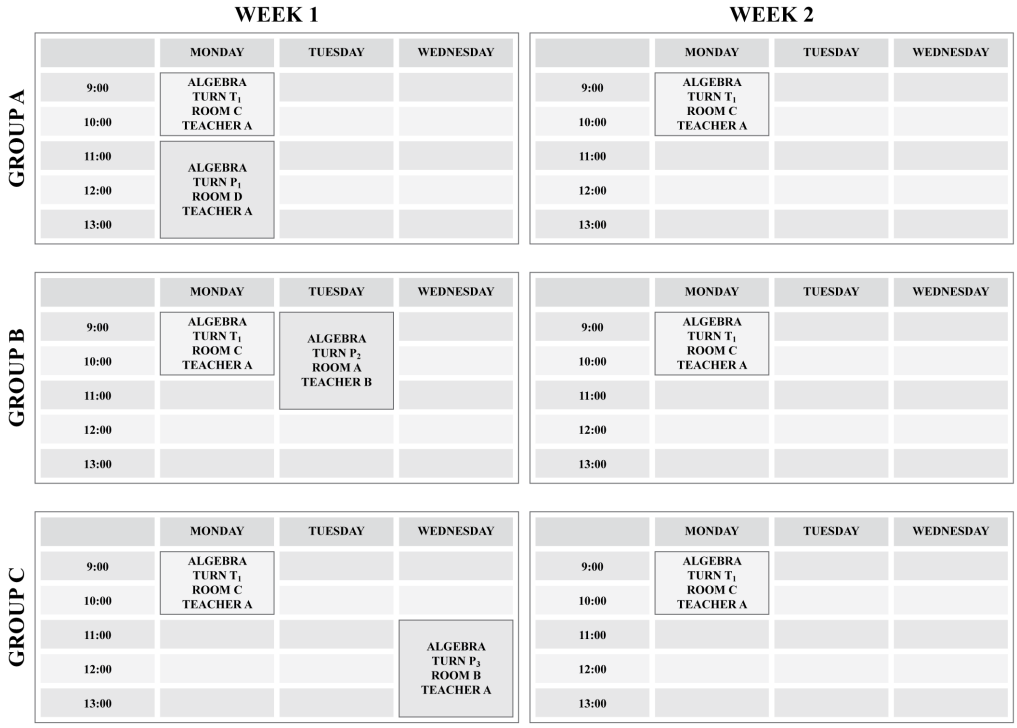
one or more **turns**. The turns represent how a particular module is taught from the point of view of the institution, that is, how many times the same lesson is repeated to the different sets of students. Turns are made up of one or more groups. Figure 2 illustrates a practical example for the specific case of the module Algebra, with two lessons (T and P), which occur on a different set of weeks: one of the theoretical type with a duration of 2 hours and the other of the practical type with a duration of 3 hours. Lesson T is composed of one turn (T_1) that aggregates the three existing groups (Group A, Group B, Group C). Lesson P is composed of three turns (P_1, P_2, P_3) with the three separate groups, one for each turn.

Figure 2 Example of the working mode of a module



Thus, to create timetables for one institution a set of **events** should be placed in a structure that keeps track of the beginning and the end of each one of them – the timesheet. Each event is composed of a turn (set of students of one or more groups), one or more classrooms and one or more teachers. An event can be taught in a predetermined classroom, in one classroom out of a set of possible classrooms or in more than one classroom simultaneously. An event can be taught by one predetermined teacher, by one teacher out of a set of possible teachers or by more than one teacher at the same time. An event will be placed in a certain position of the timesheet, with a given duration and occurring in a set of weeks, respecting all existing constraints. The timesheet should be configurable, allowing indicating the minimum slot size, that is, the possible intervals for the allocation of events (for example, if a slot of 60 minutes is defined, the timesheet will be divided into intervals of 60 minutes, and the events may be initiated in each one of such intervals). The duration of the event must be a multiple of the slot defined (in the previous example, one event of 120 minutes should be defined as having a duration equal to two slots). Figure 3 illustrates a possible placement in the timesheet of the module exemplified in Figure 2, also including the teachers (A, B) and the classrooms (A, B, C, D) of each event.

Figure 3 Example of placement of the events of one module



The introduction of a concept of multiple weeks in the creation of timetables adds a huge complexity to the problem, since the solution does not go through the "simple" creation of a base timetable for a week that repeats over the following weeks. The current trend, according to the reorganization resulting from the application of the Bologna Process, is the existence of weeks with different time patterns during a term. In order to be able to manage this new paradigm, BTTE has a data structure that allows performing in real time all necessary calculations, in order to optimize the searches and provide quality results in a timely manner.

4.2 Hard constraints

A solution is feasible only if it respects all hard constraints. There are rules specific to each institution, legal constraints, or physical impossibilities that have to be fulfilled.

Besides the natural constraints – for example, a particular resource (a teacher, a group or a classroom) can only have one event assigned to it in each period, or the use of a particular classroom is limited by its capacity – it is necessary in the Portuguese case to impose and comply with an additional set of mandatory rules.

It is important to refer that in a real situation there is a thin line that separates some situations that must be respected (hard constraints) from others that should be respected (soft constraints). Additionally, this type of classification varies from institution to institution. Thus, these cases can be found in both hard and soft constraints subchapters, since BTTE has a double application of some concepts (hard and soft), allowing the user to choose which one should be applied.

The hard constraints considered in the model adopted in the development of BTTE are presented in Table 1.

Table 1 Hard constraints

Entity	Hard Constraint	Example
Group	Conflicts	A Group can only have one event assigned in each time interval
Group	Availability	Group A cannot have events assigned on Friday
Group	Flexible pauses	Between 12h00 and 14h00, Group A must have 1 hour for meal break
Group	Maximum event load	Group A can only have a maximum of 6 hours of events each day
Group	Consecutive event load	Group A can only have a maximum of 4 consecutive hours of events each day
Group	Rest	All Groups must have a minimum rest of 12 hours between events that begin on different days
Group	Travel distance	All Groups must have at least 30 minutes for changing from Building A to Building B, between two consecutive events
Teacher	Conflicts	A Teacher can only have one event assigned in each time interval
Teacher	Availability	Teacher A cannot have events assigned on Monday morning
Teacher	Flexible pauses	Between 12h00 and 14h00, Teacher A must have 1 hour for meal break
Teacher	Maximum event load	Teacher A can only have a maximum of 6 hours of events each day
Teacher	Consecutive event load	Teacher A can only have a maximum of 4 consecutive hours of events each day
Teacher	Rest	All Teachers must have a minimum rest of 12 hours between events that begin on different days
Teacher	Travel distance	All Teachers must have at least 1 hour for changing from Building C to Building D, between two consecutive events
Teacher	Maximum working days	Teacher A can only have events assigned in 3 different days
Classroom	Conflicts	A Classroom can only have one event assigned in each time interval
Classroom	Capacity	Classroom A can only accept events with 50 or less students
Classroom	Availability	Classroom A cannot have events assigned on Wednesday
Event	Contiguous	Event A and Event B must be contiguous
Event	Different days	Event A and Event B cannot be taught on the same day
Event	Separated	Event A and Event B cannot be overlapped
Event	Overlapped	Event A and Event B must be overlapped
Institution	Availability	Institution is closed on Saturday and Sunday
Module	Availability	Events of Module A can only be assigned before 16h00
Module	Sequence	In Module A, practical events can only be taught after the theoretical ones
Turn	Availability	Turn A can only be assigned after 19h00

4.3 Soft constraints

The quality of a solution is related with the fulfilment of a set of soft constraints. The relationships between these soft constraints and the way their violation is penalized are used to define the objective function, which evaluates the quality of one complete solution.

In the Portuguese case, one can find different concepts of quality and the most diverse soft constraints that must be fulfilled so that it is possible to say that a good solution was obtained.

The soft constraints considered in the model adopted in the development of BTTE are presented in Table 2.

Table 2 Soft constraints

Entity	Soft Constraint	Example
Group	Free periods	All Groups should have at least 4 periods of the day (morning, afternoon or after work) without events assigned
Group	Minimum event load	All Groups should have at least 3 hours of events assigned per period of the day (in order to prevent them from having to travel to the institution only for a short period of time)
Group	Event types	All Groups should have events of different types (theoretical, practical) in each period of the day
Group	Gaps	The gaps between consecutive events should be minimized for all Groups
Group	Classroom stability	Classroom changes should be avoided for all Groups
Group	Travel distance	Classroom changes (different buildings) should be avoided for all Groups
Group	Availability	Group A should not have events assigned on Friday (more than one preference level may be defined)
Teacher	Free periods	All Teachers should have at least 4 periods of the day (morning, afternoon or after work) without events assigned
Teacher	Minimum event load	All Teachers should have at least 3 hours of events assigned per period of the day (in order to prevent them from having to travel to the institution only for a short period of time)
Teacher	Gaps	The gaps between consecutive events should be minimized for all Teachers
Teacher	Classroom stability	Classroom changes should be avoided for all Teachers
Teacher	Travel distance	Classroom changes (different buildings) should be avoided for all Teachers
Teacher	Availability	Teacher A should not have events assigned on Monday morning (more than one preference level may be defined)
Teacher	Maximum working days	Teacher A should only have events assigned in 3 different days
Teacher	Similar timetables	Teacher A, Teacher B and Teacher C should have events assigned on the same days. In many situations, some teachers live far away from the institution. To minimize the costs, if possible, they travel together, so it is important that they have similar timetables
Classroom	Gaps	The gaps between consecutive events should be minimized for all Classrooms
Classroom	Equipment	The alternation of events that require different equipment should be minimized for all Classrooms, to avoid logistics changes
Classroom	Suitability	The Classroom (or Classrooms) that will be assigned to each event should be the most suitable, from the point of view of utilization of its capacity or possible preference indications
Classroom	Availability	Classroom A should not have events assigned on Wednesday (more than one preference level may be defined)
Event	Linked	Event A and Event B (that can occur on different days) should begin at the same time

Event	Same classroom	Event A and Event B should be taught in the same classroom
Event	Contiguous	Event A and Event B should be contiguous
Event	Overlapped	Event A and Event B should be overlapped
Event	Period of the day	From the set of Events A, B, C, D and E, 2 of them should be assigned in the morning period, 2 in the afternoon and 1 after work
Institution	Availability	Wednesday afternoon is preferably reserved for meetings, so there should be no events on the Institution
Module	Availability	Events of Module A should only be assigned before 16h00
Module	Distribution	Events of Module A should be assigned widely separated over the week
Module	Sequence	In Module A, practical events should only be taught after the theoretical ones
Turn	Availability	Turn A should only be assigned after 19h00

The concept of quality varies from institution to institution and even within an institution. It is thus essential that the definition of the relative importance of each soft constraint is free so that it can adapt to different needs. BTTE allows the user to define the objective function.

5 Proposed algorithms

5.1 Business requirements

The creation of timetables in one Educational Institution is only complete when all events are scheduled, respecting all the existing hard constraints. One partial creation, that is, a solution that does not include all events that must be taught cannot be applied in practice. Moreover, a complete solution can be better or worse than another, respecting more or less the established soft constraints that are used to assess its final quality.

However, and not forgetting the importance of having one solution considered of good quality, in a real situation the first great difficulty is to be able to build a feasible solution to the problem, because the existing set of limitations increases every year. Without a valid solution we cannot start optimizing it; therefore, the focus should be on building the best calculation methods that allow finding an initial feasible and, preferably, robust solution. Optimizing the solution is the final step of the process of creating timetables.

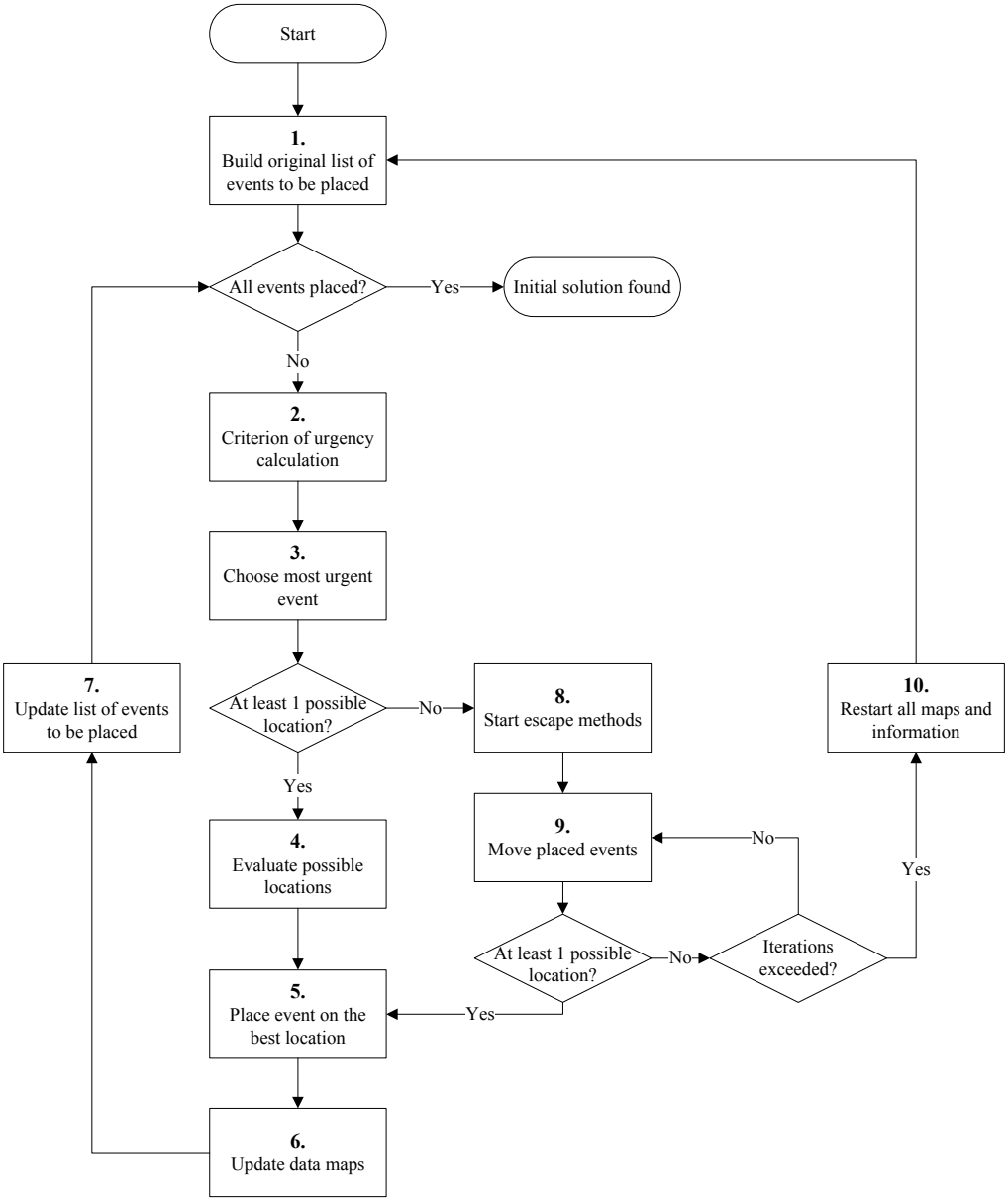
Once again, there are principles that should not be forgotten in a real situation: the final result must be presented in due time. The perception of time is usually associated with the size and complexity of the institution or the time that it needs to build manually one solution, always considering that a machine must be able to solve the problem much faster. Residual optimizations are usually considered negligible when analyzing a final solution (the user does not value one solution that requires an extra 2 hours to be reached for an outcome that is only slightly better than the previous one). It is thus necessary to support the methods of calculation at this stage by means of two conflicting principles: they must have the ability and detail to perform a deep analysis of the solution space and must have the simplicity that allows covering as quickly as possible the same space of solutions. The balance between these two factors is the key to an optimization algorithm that can be successfully applied in real cases.

Nowadays the Educational Institutions face several procedural changes, either on their own initiative, or due to external factors such as the Bologna Process. The number of constraints is ever rising, mainly due to an increased demand for optimization of human and physical resources, the need for more flexibility as well as competitive factors. This new paradigm requires frequent internal restructuring and a new way of looking at the creation of timetables, which are no more seen as something fixed and permanent over the years, but are inevitably becoming flexible, with the aim of achieving global optimizations at the institution.

5.2 Construction heuristics: building an initial timetable

A sequential heuristic is used to build an initial timetable from an empty timesheet. Figure 4 presents the construction heuristics' workflow.

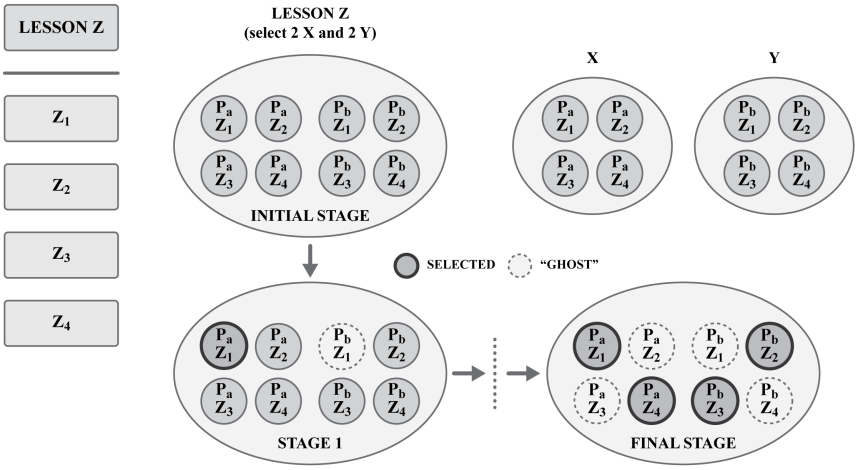
Figure 4 Construction heuristics' workflow



There is a list of all events that have not yet been placed on the timesheet (Step 1, at the starting point and Step 7, each time that a new event is placed and the list is updated). Not all elements of the list will be placed on the final timesheet, since there are "ghost" events created to allow the algorithm a greater flexibility; in this way, it is possible to use, from the outset, all existing feasible combinations. For a better understanding of this concept, the following example is given.

If Teacher P_a teaches two turns of Lesson Z (composed of turns Z_1, Z_2, Z_3, Z_4) and Teacher P_b teaches the other two turns of the same Lesson, and no other restriction exists, either teacher can teach any one of the four turns; it is only necessary to ensure that each one will teach two and only two different turns. Thus, in the list of events to place, the following tuples are possible: P_aZ_1, P_aZ_2, P_aZ_3, P_aZ_4, P_bZ_1, P_bZ_2, P_bZ_3, P_bZ_4; from the set of events of P_a (set X), two events will be placed (the other two will be transformed into "ghost" events along the way), and the same happens with the set of events of P_b (set Y). While the events are being selected for placement on the final timesheet (following the rules of the construction heuristics, that will be presented later on), the information for controlling the number of events assigned to each set is updated and other events are put "asleep" when necessary (e.g., if the event P_aZ_1 is placed, the event P_bZ_1 is no longer valid, since the turn Z_1 has already been assigned to Teacher P_a). Figure 5 exemplifies this case, from the initial stage (the starting point of the algorithm) until the final stage (when all the events are placed on the timesheet).

Figure 5 Example of using "ghost" events



Despite Figure 5 showing a straightforward example, more complex situations can occur. One teacher may already have been allocated to one specific turn or one turn may have the need of two teachers simultaneously. Using a similar structure, if Teacher P_a teaches two turns of Lesson Z (being one of them obligatorily Z_1), Teacher P_b teaches one turn, and Teachers P_c and P_d teach simultaneously one turn, the list of original possible events to be placed would be: P_aZ_1, P_aZ_2, P_aZ_3, P_aZ_4, P_bZ_2, P_bZ_3, P_bZ_4, $P_cP_dZ_2$, $P_cP_dZ_3$, $P_cP_dZ_4$. One possible final allocation would be: P_aZ_1 (always), P_aZ_2, P_bZ_3, $P_cP_dZ_4$. In this case, the remaining turns were transformed into "ghost" events along the way (for example, P_aZ_3 and $P_cP_dZ_3$ when P_bZ_3 was selected, or P_bZ_2 and $P_cP_dZ_2$ when P_aZ_2 was selected, being P_aZ_4 equally transformed in this step since Teacher P_a would already had been assigned at that moment the two turns that he must teach – P_aZ_1 and P_aZ_2).

Once a list of events for placing is defined (a complete list, including the events that will be transformed into "ghost" events along the way), it is necessary to choose which event will be placed in the timesheet. For this selection, a criterion of urgency was defined (Step 2 in Figure 4). It is calculated based on: a) the degree of dependence between events that are associated by means of constraints, since it can be extremely complex to assign sets of events that must be contiguous or overlapped (the higher, the most urgent); b) the overlapping of availability of all resources involved in the event, which indicates in how many different places the event can be assigned at each moment (the lower, the most urgent); c) the correction of the previous factor, by the elimination of potentially feasible places that are no more feasible due to other constraints than availability (the lower, the most urgent); and d) the range of possible classrooms to host the event (the lower, the most urgent).

After the selection of the most urgent event to place (Step 3), it is necessary to evaluate the best location for placing the event (Step 4). The created data structure allows evaluating only the viable places, not wasting time to analyse invalid places. The choice for the best placement is defined based on two fundamental criteria: a) the place with the highest quality, seeking to build an initial solution that already meets the soft constraints set by the user and expressed by the objective function; and b) the location that causes the least impact on the remaining events to place, seeking to avoid choices that lead to a considerable limitation of the possibilities of placing those events.

After placing one event (Step 5), all the data maps are updated (Step 6), the list of events to be placed is also updated (Step 7), the criterion of urgency is recalculated and the selection of a new most urgent event is carried out. The information structure created for BTTE allows updating and manipulating all these data maps in real time; at any time there is a clear view of the possibilities for placing each event, of its urgency and the impact of the placement of other events into all events that have not yet been placed.

Even with a careful choice of the sequence of events to process, by selecting first the most urgent ones (which are likely to be more restricted) and placing them with a criterion (causing the least impact on the locations of placement of the events that have not yet been placed), it is natural that in very restricted problems, as in almost all real cases, sometimes the event that was chosen as the most urgent has no possible place (Step 8).

After having tested, without success, traditional methods of backtracking to solve this problem, new processing methods adapted to the existing model were developed. In order to insert an event that currently has no possible location for placement, it is necessary to change something in the current solution. Assuming that if a possible solution to the problem exists then there will be at least one fitting of all events on the timesheet. This means that it is not necessary to remove events already placed (if they have been placed first, they were also restricted events), but move them to other positions, in order to try to open a new slot for the event that causes problems (Step 9). The structure of the existing information and the data maps allow, again, that these movements are not random. Information about the places originally available for the event (empty timesheet), about places that became invalid and the reasons for this to happen (common resources occupied by other events, constraints that would be broken), are undoubtedly some examples of critical data that, if well used, dramatically increase the probability of making the correct movements. It is intuitive to understand that if it was never possible to place a troublesome event on Monday (because, for example, the only Teacher who teaches this event is not available that day), attempt to release space on Monday should not be the first option to take. On the other hand, if the only obstacle is the fact that the Teacher is already being busy with another event on that day, it might be an interesting option to start trying to move him to another day or time. With these two simple and basic examples it is possible to realize the dimension and complexity required to make these escape methods "smart" enough to understand the various situations that may occur. More than understanding, they must be prepared to perform the right choices since, in the overwhelming majority of situations, the impediments are due to a combination of many factors and are not solved with simple changes as in the previous examples. This is one of the crucial aspects to deal with in order to achieve a viable solution in a real restricted environment. At present, BTTE software incorporates escape methods that are the result of several years of improvements and allow overcoming such problems. These methods are not fully explained in this paper due to the complexity involved and the detail needed to clearly present them.

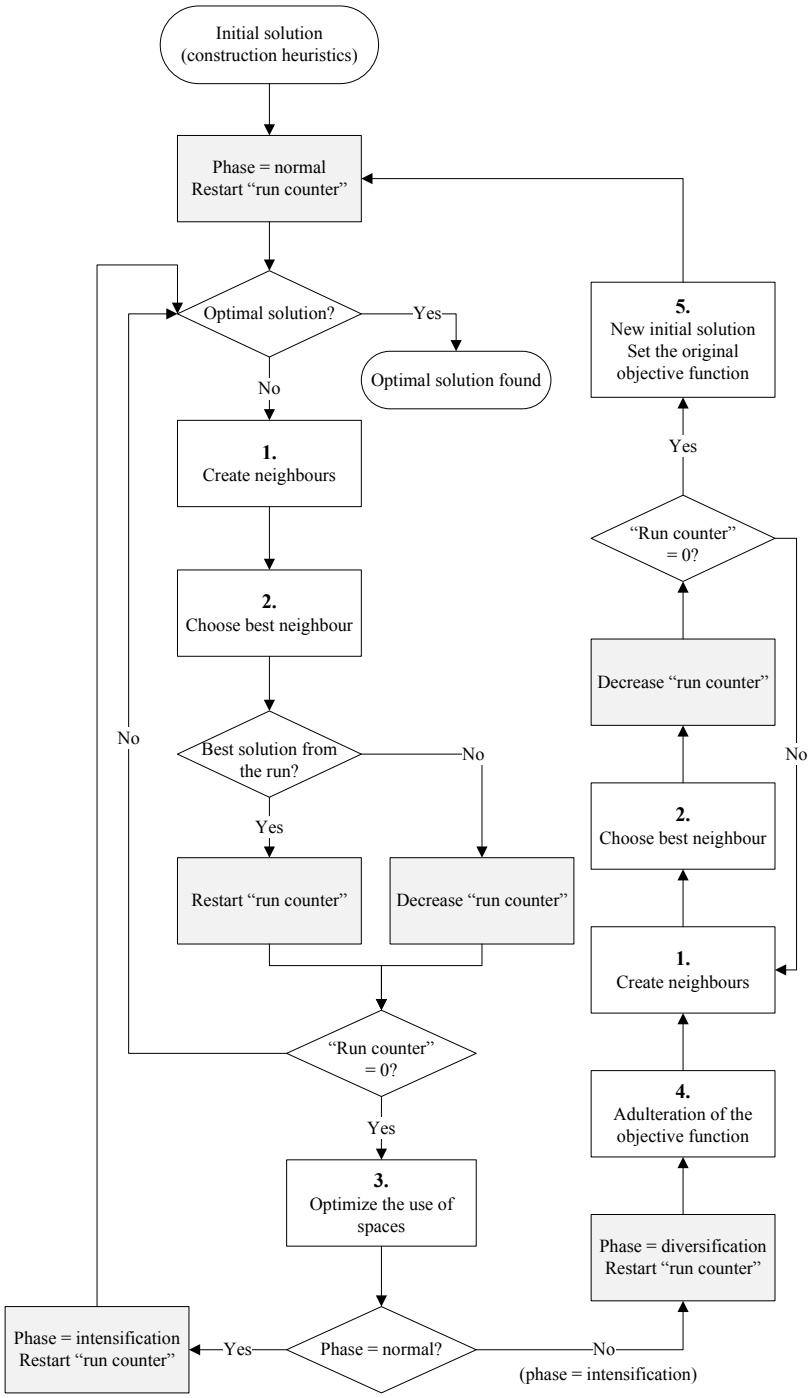
As soon as the escape methods can find a location for placing the event in trouble, the placement takes place and the construction of the initial solution proceeds normally (Step 5). If the escape methods fail to provide a feasible location for the event, after a number of consecutive iterations, a new solution is started from scratch (Step 10). At the same time, the user is also advised to review the constraints imposed to the program, with the help of some reports produced by BTTE.

The initial solution is found when all events have been successfully placed.

5.3 Improvement heuristics: finding a final timetable

Once the initial solution to the problem is found (the starting point), the optimization phase is initiated; based on appropriate methods, better solutions are progressively searched. Figure 6 presents a simplified version of the improvement heuristics' workflow.

Figure 6 Improvement heuristics' workflow



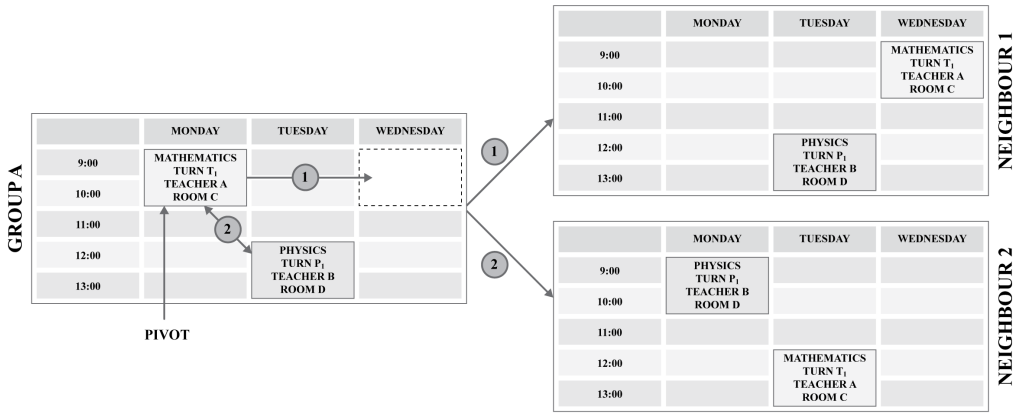
In BTTE, the search for new solutions is based on neighbourhood structures, that is, an iterative construction of sets of neighbours (solutions with small changes compared to the current one) that are evaluated and accepted (or not) according to specific parameters, thereby allowing an exploration of the solution space. Besides the different construction methods of neighbourhoods, the optimization algorithms implemented go through three major phases: normal, intensification and diversification. Each of these phases is specified to achieve a particular purpose, and their joint operation is the key to a final optimized outcome. The heuristics was fully developed from scratch, adapted to the existing problem and created model, but concepts and principles of meta-heuristics such as the Tabu Search [9] served as the basis for the research conducted.

The option of using more than one neighbourhood structure allowed diversifying the search, making it more robust and complete. Thus, two different modes were implemented to create neighbours, that is, new solutions (Step 1 in Figure 6): direct exchanges between events and exchanges between events in two steps.

In the first case, a pivot event is chosen and, with the support of existing data maps, potential destination places are selected. These places can be free or occupied. If they are free, there is immediately a valid neighbour. If they are busy, the event or events that are occupying the destination place will be moved to the original location of the pivot event, thus creating a valid neighbour if the exchange is possible.

Figure 7 illustrates the first neighbourhood structure, using two events of one group.

Figure 7 First neighbourhood structure



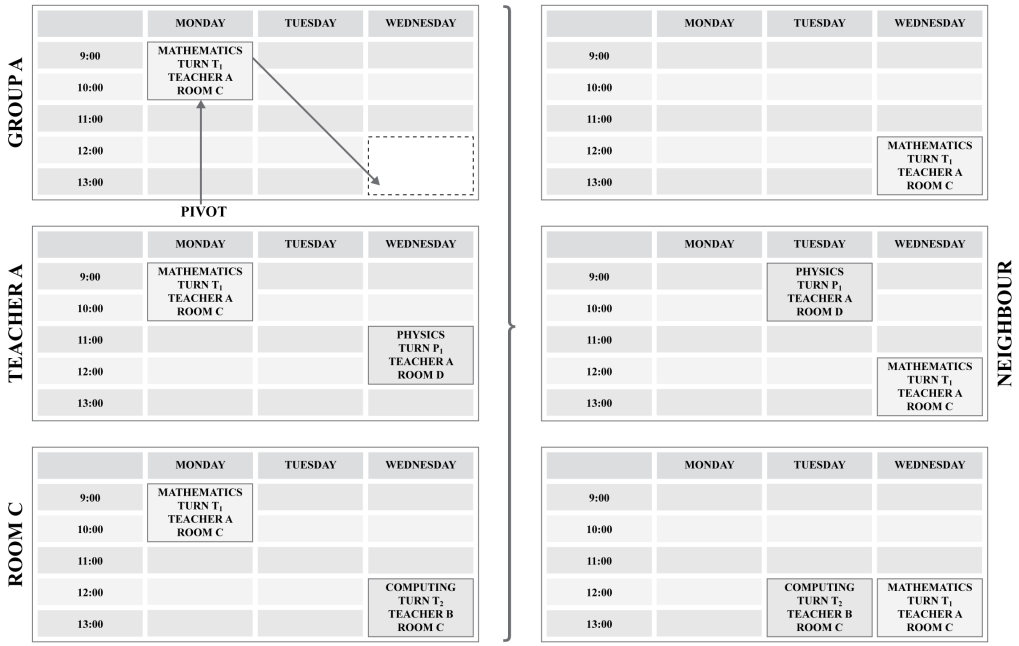
In the second case, the main difference is in the way the events that occupy the destination place are treated. A more complete assessment of existing overlaps is performed and alternative locations for placement, besides the original site of the pivot event, are looked for.

Figure 8 illustrates the second neighbourhood structure, using three events and entities (group, teacher and classroom) associated with the pivot event.

With the first neighbourhood structure, it is possible to achieve results faster and with fewer exchanges in each iteration (the neighbours are more "similar", since they are constructed based on direct exchanges between events or movements to free places).

With the second neighbourhood structure it is possible to find new feasible and interesting neighbours with only one iteration (the neighbours may have major differences, since both the number of events included in the exchange and the potential destination places for each one can be higher).

Figure 8 Second neighbourhood structure



In each iteration, the existing best neighbour (the one with the best value of the objective function) is always accepted (Step 2 in Figure 6), even if this neighbour has a lower quality than the current solution. After several tests, it was decided to adopt this approach, since in the vast majority of situations this acceptance technique allowed escaping from local minima and not endangering the fast return to quality solutions.

Depending upon the search phase (normal, intensification or diversification), the use of the described neighbourhoods has specific characteristics.

In the normal phase, the goal is performing a faster search and a more free exploration of the solution space. Thus, the number of neighbours created in each iteration is limited, either by reducing the number of pivot events used or by reducing the number of potential destination places to be evaluated.

In the intensification phase, the most promising search zones are explored in more detail. At this stage, the algorithm returns to the solutions of better quality that were found in the normal phase, performing a deeper search around it (increasing the number of pivot events used and evaluating the whole set of potential destination places for each one of them).

The stopping criterion in each one of these phases is applied after a number of consecutive iterations without improving the current solution. It is then assumed that a deeper search near the best solutions should be performed (moving from the normal to the intensification phase), or the search should be directed to another zone of the solution space (moving from the intensification to the diversification phase).

At the diversification stage, by means of the temporary adulteration of the objective function during a given number of iterations (Step 4), it is possible to simulate a new starting point for a new search (Step 5), without losing time needed to create a new solution from the scratch and without losing too much quality, since the objective function is only slightly changed and is not ignored.

In addition, an algorithm to optimize the use of spaces was developed (Step 3); it is focused on the soft constraints related with the classrooms used by events. In this algorithm, the events are not moved from their places and there is only a reallocation of the classrooms to

them, trying to optimize their use and the impact on other entities involved. Since it is only necessary to evaluate the soft constraints related to the classrooms, keeping all the remaining unchanged, the required improvements are reached very quickly. This algorithm runs either when going from the normal to the intensification phase or from the intensification to the diversification phase. The above-mentioned stop criterion is kept, that is, the algorithm is active until a certain number of consecutive iterations without improving the current solution is reached.

The optimization process may end on request by the user and, at that time, the best solution found so far is returned. It can also end automatically in case the optimal solution to the problem is found ("zero" penalty of the objective function). This is nevertheless highly improbable in a real scenario, since many of the used soft constraints are competitors.

6 Case studies

6.1 Usual process of building up the timetables

To achieve a final result, there are a few sequential steps that are usually followed. First, the information is loaded into the system; in this step, normally some initial adjustments are made. Then, an initial solution is calculated. With a complete solution in hands (but not yet optimized), the timetabling team searches for data errors or information inconsistencies, and a new round of adjustments is performed. These steps are repeated until the data is considered stable. The last step is producing a final timetable, where the initial solution is calculated one more time and then optimized.

When the automatic generation stage ends, BTTE sends the timetables to another software module, Bullet Calendar (BC), where the user can edit timetables and manage events and resources.

BC is an add-on module to BTTE, which allows an efficient daily management of all activities and resources in an Educational Institution, through a support decision tool.

BTTE is the intelligent centrepiece of the system, the calculation engine and therefore it is the main focus of this paper. However, BC is a module that is used continually, since it can easily respond to all the changes that inevitably occur during the year. If, at any time, profound changes are needed, the user can get back to BTTE and generate a complete solution from scratch, or just generate the affected events keeping the rest of the solution untouched in BC.

Some of the main features of BC are: totally flexible agenda, with easy configuration of time slots and working hours; real-time information about the occupation of the Institution's resources; quick editing of timetables with several views; quick analysis of all the possibilities and limitations of swapping a specific event; total flexibility for changing any resource associated with any event; fast Web publishing of timetables; exporting and printing schedules and dozens of different reports.

6.2 Description and main features

Twenty real cases of Portuguese institutions, users of BTTE, were analyzed. The selected cases were as heterogeneous as possible, with different degrees of complexity and dimension, in order to have a good representation of the Portuguese reality.

The case studies, presented in Table 3, can be classified by means of 11 critical factors (CF) that allow introducing complexity into the problem: 1) number of different weeks (the higher, the more complex); 2) existence of contiguous and overlapped events (the higher, the more complex); 3) number of events to place (the higher, the more complex); 4) number of groups (the higher, the more complex); 5) number of teachers (the higher, the more complex); 6) existence of availability constraints of resources (the higher, the more complex); 7) overlapping of regimes (more complex); 8) existence of high competition for physical

resources (the higher, the more complex); 9) existence of high diversity in the aggregation of groups to make up turns (the higher, the more complex); 10) size of the slot on the timesheet, in minutes (the smaller, the more complex); 11) number of soft constraints to analyze (the higher, the more complex).

Table 3 Case studies and critical factors that allow introducing complexity into the problem

	CF1	CF2	CF3	CF4	CF5	CF6	CF7	CF8	CF9	CF10	CF11
Case 1	2	none	144	28	84	medium	no	low	low	30	21
Case 2	1	none	341	50	100	low	no	medium	low	60	16
Case 3	1	none	90	55	92	medium	yes	medium	medium	30	15
Case 4	1	none	169	38	75	low	no	low	low	60	15
Case 5	1	none	130	17	34	low	no	low	low	30	17
Case 6	2	none	200	39	83	medium	yes	low	low	15	16
Case 7	4	few	485	52	117	high	yes	high	low	30	16
Case 8	1	none	350	350	150	high	no	high	high	5	15
Case 9	2	none	478	64	161	low	no	low	low	90	16
Case 10	1	none	453	83	139	high	yes	medium	low	60	15
Case 11	1	few	650	70	100	low	no	medium	medium	90	18
Case 12	1	none	200	47	101	medium	yes	medium	low	30	16
Case 13	1	few	532	94	362	high	no	high	high	30	16
Case 14	2	few	605	205	667	high	yes	medium	medium	30	19
Case 15	2	none	1200	127	621	medium	yes	medium	low	30	17
Case 16	1	none	1350	185	532	low	no	medium	low	30	18
Case 17	4	few	1800	239	512	medium	no	medium	low	30	19
Case 18	13	many	1855	825	600	medium	no	high	medium	30	16
Case 19	12	none	1985	300	471	low	no	high	low	30	15
Case 20	15	none	3500	1346	1250	medium	yes	medium	medium	30	17

6.3 Results and discussion

Since the model used by Bullet Solutions is not found in any publication in the area, it is not possible to perform a comparison with other studies. The option was thus to present the significant differences found between the previous manual process of creating timetables for each one of the selected institutions and the results they now can achieve with BTTE.

The time required for BTTE to provide the final optimized results can be divided into three parts: 1) average time required to load and/or validate the information in the system (TVBTTE); 2) average time needed for calculating the initial solution (TISBTTE); and 3) average time required for the calculation of an optimized solution (TOBTTE). The sum of these times represents the total utilization time of the product (TTBTTE), which can be compared with the total time that was spent in the previous manual process (TTMP). TTMP was provided by the timetabling team of each institution.

The machine used to perform the simulations was an Intel Core 2 Duo 2.53 GHz, with 4 GB of RAM. Twenty simulations were performed for each of the twenty case studies. The results (average values) are presented in Table 4.

Table 4 Case studies and average times associated with the process of creating timetables

	TVBTTE (hh:mm:ss)	TISBTTE (hh:mm:ss)	TOBTTE (hh:mm:ss)	TTBTTE (hh:mm:ss)	TTMP (hh:mm:ss)	Savings (hours)	Savings (%)
Case 1	4:30:00	0:00:45	0:30:00	5:00:45	48:00:00	43	89,56
Case 2	11:00:00	0:01:00	0:40:00	11:41:00	130:00:00	118	91,01
Case 3	3:00:00	0:00:30	0:30:00	3:30:30	32:00:00	28	89,04
Case 4	5:30:00	0:00:15	0:15:00	5:45:15	72:00:00	66	92,01
Case 5	5:00:00	0:00:05	0:45:00	5:45:05	48:00:00	42	88,02
Case 6	6:30:00	0:01:00	1:00:00	7:31:00	72:00:00	64	89,56
Case 7	16:00:00	0:05:00	2:00:00	18:05:00	120:00:00	102	84,93
Case 8	12:00:00	1:00:00	12:00:00	25:00:00	144:00:00	119	82,64
Case 9	10:00:00	0:01:00	0:40:00	10:41:00	192:00:00	181	94,44
Case 10	24:00:00	0:02:00	0:30:00	24:32:00	168:00:00	143	85,40
Case 11	22:00:00	0:04:00	2:00:00	24:04:00	216:00:00	192	88,86
Case 12	8:00:00	0:01:00	1:00:00	9:01:00	72:00:00	63	87,48
Case 13	18:00:00	0:01:00	2:00:00	20:01:00	192:00:00	172	89,57
Case 14	22:00:00	0:10:00	2:00:00	24:10:00	288:00:00	264	91,61
Case 15	38:00:00	0:15:00	5:00:00	43:15:00	408:00:00	365	89,40
Case 16	44:00:00	0:45:00	8:00:00	52:45:00	504:00:00	451	89,53
Case 17	50:00:00	1:00:00	10:00:00	61:00:00	432:00:00	371	85,88
Case 18	60:00:00	8:00:00	18:00:00	86:00:00	528:00:00	442	83,71
Case 19	80:00:00	2:00:00	12:00:00	94:00:00	408:00:00	314	76,96
Case 20	120:00:00	4:00:00	120:00:00	244:00:00	936:00:00	692	73,93

In the twenty cases studied, the opinion of the users was that BTTE produced results of better quality when compared with the timetables that were created manually. Similarly, the huge timesaving (above 85%, on average) and the reduction of the resources involved in the process allowed the institutions to face the problem of creating timetables in a much safer way.

From the analysis of information and results presented in the tables, some cases should be highlighted. Case 8 seems rather simple, but generation times are considerably higher than those of other seemingly similar cases. The truth is that when using a slot of 5 minutes in the timesheet (CF10) and having a high diversity in the aggregation of groups to constitute turns (CF9), the processing complexity increases immediately. The significant increase of time required for optimization in Case 15 and beyond (up to Case 20) is explained, at first, by the number of events that exist in the problem (CF3), well above the 14 previous examples. Cases 18, 19 and 20 involve a number of different weeks (CF1) much higher than the others; Case 18 introduces a very specific additional difficulty related with the large number of hard constraints of contiguous and overlapped events (CF2). Case 20 is the most complete and complex of all, with an enormous variety of factors that have an impact on the calculation of the solution.

6 Summary and conclusions

The main conclusions that may be extracted from the work developed are intimately connected with the commercial success of the BTTE application. Only with a correct modelling of the problem (considering that “correct” means that a user sees it as a mirror of reality) it is

possible to create the information base necessary for the process of creation of timetables, as well as all existing relationships. Only with robust algorithms, it is possible to create valid solutions to problems with so many restrictions as those found in almost all Educational Institutions. Only with additional methods of optimization it is possible to find results that meet users' expectations, both in what concerns the quality presented and the speed to achieve them (the user always expects that the machine will be able to reach a better result and much faster than a manual solution).

The fact that results with good quality, with savings of 85% on the time spent in the process, were obtained in all analyzed cases, allows concluding that the algorithms used in the BTTE application have a considerable level of robustness and ease of adaptation to the quite diverse real scenarios that were used for their evaluation.

The implementation and use of the BTTE application allowed significant improvements in processes directly and indirectly related with the creation of timetables, resulting in additional productivity gains.

It was possible to observe the organization and centralization of information in most of the institutions and the elimination of redundant information. An increase in speed and efficiency in the workflow and the information flow was observed too.

Some solutions for consulting in real time the timetables produced were also implemented, thus meaning that these institutions are well aligned with the Digital Society procedures.

At present, there is an improvement in the image and positioning of some institutions in the market, by using advanced technology for automation and optimization of processes.

There is a better control of top management staff over the real needs of the distribution of teaching service. Scenarios in which the generation of timetables was far from top management and was fragmented by several departments of the institution were common in the past, resulting in recruitment of resources that later on revealed to be not necessary. There have been considerable savings in hiring teachers, after using the BTTE application.

Acknowledgements The authors wish to thank the entire team of Bullet Solutions who participated over the years on the project of creating the BTTE software. The authors also thank all Educational Institutions that have contributed to make it possible, in the first instance to create a realistic model of the problem and, in a second phase, to make the application increasingly robust and comprehensive. Only with the contribution of all these people and entities it was possible to develop a product that currently solves the problem of creating timetables for Portuguese Educational Institutions.

References

1. A. Schaerf, “A Survey of Automated Timetabling”, *Artificial Intelligence Review*, vol. 13, no. 2, pp. 87–127 (1999)
2. A. Bonutti, F. De Cesco, L. Di Gaspero and A. Schaerf, “Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results”, *Annals of Operations Research*, vol. 194, no. 1, pp. 59–70 (2012)
3. R. Lewis, “A survey of metaheuristic-based techniques for University Timetabling problems”, *OR Spectrum*, vol. 30, no. 1, pp. 167–190 (2007)
4. R. Alvarez-Valdes, E. Crespo and J. M. Tamarit, “Design and implementation of a course scheduling system using Tabu Search”, *European Journal of Operational Research*, vol. 137, no. 3, pp. 512–523 (2002)
5. K. Murray, T. Müller and H. Rudová, “Modeling and Solution of a Complex University Course Timetabling Problem”, *Practice and Theory of Automated Timetabling VI*, E. K. Burke and H. Rudová, Eds. Springer-Verlag Berlin, pp. 189–209 (2007)

6. B. McCollum, “University Timetabling: Bridging the Gap between Research and Practice”, in *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, pp. 15–35 (2006)
7. S. Daskalaki and T. Birbas, “Efficient solutions for a university timetabling problem through integer programming”, *European Journal of Operational Research*, vol. 160, no. 1, pp. 106–120 (2005)
8. S. Daskalaki, T. Birbas and E. Housos, “An integer programming formulation for a case study in university timetabling”, *European Journal of Operational Research*, vol. 153, no. 1, pp. 117–135 (2004)
9. C. Blum and A. Roli, “Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison”, *ACM Computing Surveys (CSUR)*, vol. 35, no. 3, pp. 268–308 (2003)
10. S. Abdullah, E. K. Burke and B. McCollum, “An investigation of variable neighbourhood search for university course timetabling”, in *The 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, pp. 413–427 (2005)
11. R. K. Ahuja, Ö. Ergun, J. B. Orlin and A. P. Punnen, “A survey of very large-scale neighborhood search techniques”, *Discrete Applied Mathematics*, vol. 123, no. 1–3, pp. 75–102 (2002)
12. M. W. Carter, “A Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo”, in *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling (selected paper),* Springer-Verlag London, pp. 64–84 (2001)
13. G. Post, S. Ahmadi, S. Daskalaki, J. H. Kingston, J. Kyngas, C. Nurmi and D. Ranson, “An XML format for benchmarks in High School Timetabling”, *Annals of Operations Research*, vol. 194, no. 1, pp. 385–397 (2012)

MISTA 2013

The single-machine scheduling problem with a finite capacity input buffer

Shunji Tanaka

Abstract This study treats the single-machine scheduling problem with a finite capacity input buffer. In this problem, each job is assigned to the machine at its release date and started immediately, or, is sent to the input buffer to wait for processing. The problem in this study is to find a schedule that minimizes the required size of the input buffer. Its strong NP-hardness will be shown first and next, an integer programming formulation will be presented. Another integer programming formulation will also be derived for the special case when the sizes of the jobs are the same. Then, a simple heuristic algorithm will be proposed. Finally, the effectiveness of the formulations and the heuristic algorithm will be examined by numerical experiments.

1 Introduction

This study treats the single-machine scheduling problem with a finite capacity input buffer. Suppose that n jobs (job 1, ..., job n) are to be processed on a single machine. Each job i is given a processing time p_i, a size b_i and a release date r_i. When job i arrives at r_i, it is either started immediately on the machine without preemption, or kept waiting in the input buffer until it is processed. To store job i, a b_i space is required of the buffer. The problem in this study is to find an optimal schedule, i.e. an optimal processing order of the jobs on the machine (Fig. 1). Among various objectives that are possibly considered, the input buffer size is minimized in this study.

This problem typically appears when we focus on a single machine of a multi-stage processing line. Moreover, the following scheduling problem in a small-scale distribution center is modeled as this problem. Suppose that truck i arrives at the distribution center at time r_i. Each truck i should load and unload items there, which takes a p_i time. Only one truck can be processed at a time and preemption is not allowed. There is a parking area where trucks can wait for processing by occupying a b_i space (the size of truck i is given by b_i). Then, the scheduling problem in this distribution center is treated as the aforementioned scheduling problem. These examples imply that the framework in this study is applicable to a wide class of practical problems.

Shunji Tanaka
Center for Global Leadership Engineering Education, Kyoto University
E-mail: tanaka@kuee.kyoto-u.ac.jp

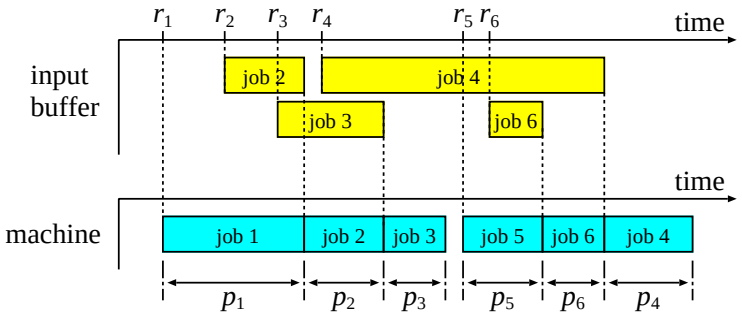


Fig. 1 The Single-Machine Scheduling Problem with a Finite Capacity Input Buffer

There have been several studies on the single-machine scheduling problem with a finite capacity input buffer. [1–5,7,9,8]. In the problems considered by the earlier works [1–5], all the jobs are not necessarily processed. The only exception is [4]; however, not an input buffer but an output buffer was considered in the single-machine scheduling problem. It is true that there are some similarities between the problem with an input buffer and that with an output buffer, but the buffer size minimization problem was not considered in [4]. In the recent studies [7,9], more complicated models with an output buffer (inventory) were considered where "positive jobs" increase the inventory level when they are finished, while "negative jobs" decrease it. Another model also with an output buffer was treated in [8], where several complexity results were shown.

The purpose of this study is to treat the single-machine scheduling problem with an input buffer under the constraint that all the jobs should be processed. More specifically, the following problem (P) is solved.

(P) Given n jobs are to be processed on a single machine. The machine can process at most one job at a time and no preemption is allowed. For each job i, an integer release date r_i, an integer processing time p_i and an integer size b_i are given in advance. It is assumed without loss of generality that $0 = r_1 \leq r_2 \leq \cdots \leq r_n$. Job i that arrives at r_i is immediately started on the machine, or kept waiting for processing in the input buffer. The size of job i is b_i and job i occupies a b_i space when it is waiting in the input buffer. The problem is to find an optimal processing order of the jobs that minimizes the required buffer size.

In this study, the strong NP-hardness of (P) will be shown first. Next, an integer programming formulation will be given to solve the problem. Another formulation will be derived for the unit size problem, i.e. (P) with $b_i \equiv 1$. Then, a simple heuristics will be constructed, and finally the effectiveness of the formulations and the heuristics will be examined by numerical experiments.

2 Computational Complexity

First, the following theorem will be proved.

Theorem 1 *(P) is NP-hard in the strong sense even when* $b_i \equiv 1$.

Proof The outline of the proof is as follows. First, the decision problem (D) of (P) is considered as follows.

(D) Is there a feasible solution such that the required buffer size is less than or equal to a given constant B?

Since (D) belongs to NP, it suffices to prove that there exists a pseudo-polynomial time reduction from a strongly NP-complete problem to an instance of (D). For this purpose, the 3-partition problem [10] is considered.

3-PARTITION[10] Suppose that $3m$ positive integers $a_1, \ldots, a_{3m}$ satisfy

$$\sum_{i=1}^{3m} a_i = mA, \tag{1}$$

and

$$\frac{A}{4} < a_i < \frac{A}{2}, \quad 1 \le i \le 3m, \tag{2}$$

for some integer A. Can the set $\{1, \ldots, 3m\}$ be partitioned into m subsets $S_1, \ldots, S_m$ such that S_j has exactly three elements and satisfies the following equation?

$$\sum_{i \in S_j} a_i = A, \quad 1 \le j \le m. \tag{3}$$

Assume without loss of generality that A is a multiple of 3. (If not, A is multiplied by 3, which does not affect the following complexity result.) Under this assumption, the instance (I) of (D) is defined as follows.

$$B = 3m - 1, \tag{4}$$

$$n = 6m + g_{m-1} = \frac{1}{2}(3m^2 + 11m - 2), \tag{5}$$

$$r_i = \begin{cases} 0, & 1 \le i \le 3m, \\ h_k, & 3m + g_{k-1} + 1 \le i \le 3m + g_k - 1,\ 1 \le k \le m-1, \\ h_k + \frac{A}{3}, & i = 3m + g_k,\ 1 \le k \le m-1, \\ h_m, & 3m + g_{m-1} + 1 \le i \le 6m + g_{m-1}, \end{cases} \tag{6}$$

$$p_i = \begin{cases} \frac{A}{3} + a_i, & 1 \le i \le 3m, \\ \frac{A}{3}, & 3m + 1 \le i \le n, \end{cases} \tag{7}$$

$$b_i = 1,\ 1 \le i \le n, \tag{8}$$

where g_k $(0 \le k \le m-1)$ and h_k $(1 \le k \le m)$ are given by

$$g_k = \sum_{j=1}^{k} (3j+1) = \frac{1}{2}k(3k+5), \tag{9}$$

$$h_k = \frac{A}{3} g_{k-1} + 2Ak = \frac{A}{6}(3k^2 + 11k - 2). \tag{10}$$

Then, what should be proved here is the equivalence of the feasibility of 3-PARTITION and (I). Hereafter, the subsets J_k $(0 \le k \le m)$ are defined by

$$J_k = \begin{cases} \{1, \ldots, 3m\}, & k = 0, \\ \{3m + g_{k-1} + 1, \ldots, 3m + g_k\}, & 1 \le k \le m-1, \\ \{3m + g_{m-1} + 1, \ldots, 6m + g_{m-1}\}, & k = m. \end{cases} \tag{11}$$

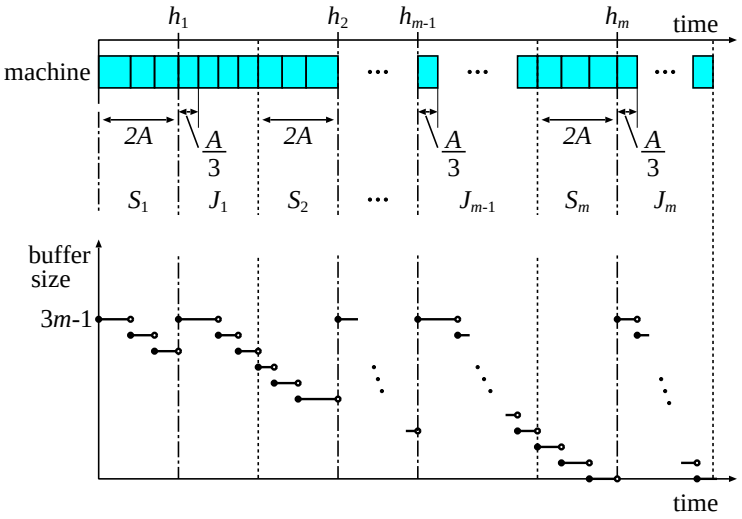


Fig. 2 Feasible Solution of (I) Constructed from Solution of 3-PARTITION

($\Rightarrow$) Assume that 3-PARTITION has a feasible solution, where S_k $(1 \le k \le m)$ are given without loss of generality by

$$S_k = \{3k-2, 3k-1, 3k\}, \quad 1 \le k \le m. \tag{12}$$

Then, a feasible solution of (I) can be constructed by choosing the completion time C_i of job i as follows (see Fig. 2):

$$C_i = \begin{cases} h_k - \sum_{j=i+1}^{3k} p_j, & 3k-2 \le i \le 3k,\ 1 \le k \le m, \\ h_k + \frac{A}{3}(i - 3m - g_{k-1}), & 3m + g_{k-1} + 1 \le i \le 3m + g_k,\ 1 \le k \le m-1, \\ h_m + \frac{A}{3}(i - 3m - g_{m-1}), & 3m + g_{m-1} + 1 \le i \le 6m + g_{m-1}. \end{cases} \tag{13}$$

($\Leftarrow$) Conversely, let us assume that (I) has a feasible solution. From $|J_m| = 3m$ and

$$r_i = h_m, \quad i \in J_m, \tag{14}$$

all the jobs in $J_0 \cup \cdots \cup J_{m-1}$ should be finished until h_m. Otherwise, the occupied space in the input buffer, i.e. the number of jobs in the buffer exceeds $3m-1$ at time h_m. Moreover, the jobs in $J_0 \cup \cdots \cup J_{m-1}$ should be processed without idle time because

$$\sum_{i \in J_0 \cup \cdots \cup J_{m-1}} p_i = h_m. \tag{15}$$

Furthermore, a job can be interchanged with the immediately preceding job without increasing the number of jobs in the buffer, if the processing time of the preceding job is shorter than that of the job and, at the same time, the interchange does not yield an idle time. Therefore, we can without loss of optimality restrict our attention on those schedules where a job with a shorter processing time is processed as earlier as possible. In these schedules, the jobs in J_0 are separated by the jobs in $J_1 \cup \cdots \cup J_{m-1}$ as shown in Fig. 3 because the former have longer processing times.

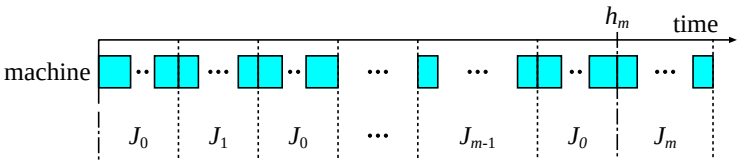


Fig. 3 Feasible Solution of (I)

First, let us consider the number of jobs in the buffer at $h_1(=2A)$. Since the number of the jobs with $r_i = 0$ is $3m$ and three among the four jobs in J_1 arrive at h_1, the number of jobs in the buffer at h_1 is given by

$$3m+3-(\text{the number of jobs already finished or under processing at } h_1). \tag{16}$$

To satisfy $B \leq 3m-1$, more than three jobs should be started until h_1 (including the job started at h_1). Among them, let us denote by J_0^1 those belonging to J_0. Then, from (2) and (8),

$$(\frac{A}{3}+\frac{A}{4})(|J_0^1|-1) < h_1 = 2A. \tag{17}$$

It implies that $|J_0^1| \leq 4$. This relation together with the fact that at most one job can be started at h_1 yields that $|J_0^1|$ is 3 or 4.

Assume that $|J_0^1| = 4$. In this case,

$$\sum_{i \in J_0^1} p_i > 4(\frac{A}{3}+\frac{A}{4}) = \frac{7A}{3} = h_1 + \frac{A}{3} = r_{3m+4}. \tag{18}$$

Therefore, not all the jobs in J_0^1 can be finished before r_{3m+4}. It follows that the starting times of the four jobs in J_1 are after r_{3m+4} ($> r_{3m+1} = r_{3m+2} = r_{3m+3} = h_1$) and thus the number of jobs in the input buffer at r_{3m+4} is $3m-4+4 = 3m > B$. Hence, $|J_0^1|$ should be 3. In this case, a job in J_1 is started at h_1. Since no idle time occurs in the schedule, the total processing time of the jobs in J_0^1 should be equal to $h_1 = 2A$. This yields

$$\sum_{i \in J_0^1} a_i = A. \tag{19}$$

By applying the same discussions to the later jobs, it can be proved that the jobs in J_0 are separated by $J_1, \ldots, J_{m-1}$ into subsets with three jobs, and that the total processing time of jobs in each subset is equal to $2A$. In conclusion, these subsets give a feasible solution of 3-PARTITION.

3 Integer Programming Formulation

In this section, (P) will be formulated as an integer programming problem.

Let us introduce the decision variables x_{it} ($1 \leq i \leq n$, $r_i \leq t \leq r_n$) so that $x_{it} = 1$ when job i is started at t (the completion time C_i of job i is $C_i = t + p_i$), and $x_{it} = 0$ otherwise. Let us also introduce $b_{\max}$, the maximum occupied space in the buffer. Next, the number of distinct release dates is denoted by n_{R}, and their sequence in increasing order by $r'_1, \ldots, r'_{n_{\mathrm{R}}}$.

Furthermore, B_k ($1 \leq k \leq n_{\mathrm{R}}$) is defined by $B_k = \sum_{\substack{1 \leq i \leq n \\ r_i \leq r'_k}} b_i$. Then, (P) can be formulated as the following integer programming problem (IP):

$$\min \ b_{\max}, \tag{20}$$

$$\text{s.t. } b_{\max} \geq B_k - \sum_{i=1}^{n} b_i \sum_{t=r_i}^{r'_k} x_{it}, \quad 1 \leq k \leq n_{\mathrm{R}}, \tag{21}$$

$$\sum_{t=r_i}^{r_n} x_{it} \leq 1, \quad 1 \leq i \leq n, \tag{22}$$

$$\sum_{i=1}^{n} \sum_{s=\max(t-p_i+1, r_i)}^{t} x_{is} \leq 1, \quad r_1 \leq t \leq r_n, \tag{23}$$

$$x_{it} \in \{0, 1\}, \quad 1 \leq i \leq n, \ r_i \leq t \leq r_n, \tag{24}$$

$$b_{\max} \in \mathbb{Z}. \tag{25}$$

In this formulation, the righthand side of (21) denotes the occupied space in the buffer at r'_k. In addition, (22) denotes the constraints that job i can be processed at most once until $r_n (= r'_{n_{\mathrm{R}}})$. Since it is possible that some jobs are processed after r_n, it is an inequality constraint instead of an equality one. In other words, this formulation only takes into account the interval $[0, r_n]$. It is because the schedule after r_n does not affect the maximum buffer size. Finally, (23) denotes the constraints that at most one job can be processed at t.

4 Another Integer Programming Formulation for the Unit Size Problem

Although we can solve (P) by applying a general MIP solver to (IP), it takes a long computation time when r_n is large because the number of decision variables is $O(nr_n)$. Therefore, another formulation for the unit size problem ((P) with $b_i \equiv 1$) will be presented. In the following, (P) with $b_i \equiv 1$ will be converted into an equivalent problem (P'), and it will be then formulated as (IP').

4.1 Conversion to Equivalent Problem

First, (P) with $b_i \equiv 1$ is converted into (P'). In (P), a job under processing is not in the buffer but on the machine, while in (P'), a job is not removed from the buffer until its processing is finished (see Fig. 4). It corresponds to assuming that the machine moves inside the buffer to process jobs (e.g. a crane).

The problem (P') is to minimize the required buffer size under this setting. Let us denote the minimum required buffer size by b'_{opt}. Then, the following theorem relates b'_{opt} with the minimum required buffer size b_{opt} of (P) with $b_i \equiv 1$.

Theorem 2

$$b'_{\mathrm{opt}} = b_{\mathrm{opt}} + 1 \tag{26}$$

Proof Let us denote an optimal solution of (P) with $b_i \equiv 1$ by $\mathscr{S}$, and its objective value by b_{opt}. Let us also denote an optimal solution of (P') by $\mathscr{S}'$ and its objective value by $b_{\mathrm{opt}} + 1$. First, it will be shown that $\mathscr{S}$ can be converted to $\mathscr{S}'$. By regarding the machine as a part

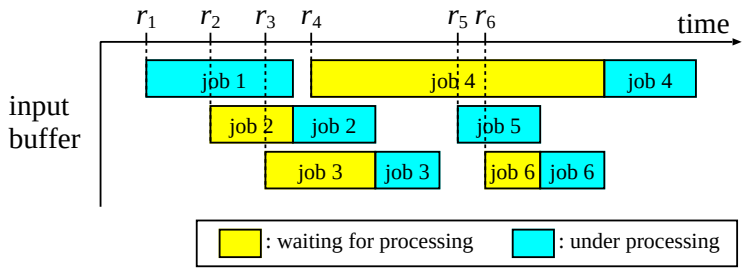


Fig. 4 Equivalent Problem (P')

of the buffer, in other words, assuming that jobs under processing are not removed from the buffer, we can obtain a feasible solution of (P'). Clearly, its objective value is either b_{opt} or $b_{\text{opt}}+1$ because the machine can process at most one job at a time. Assume that the objective value is b_{opt}. In this case, it is necessary that the occupied buffer space (the number of jobs in the buffer) in $\mathscr{S}$ should not be greater than $b_{\text{opt}}-1$ when some job is being processed. In other words, the number of jobs in $\mathscr{S}$ can be b_{opt} only when no job is being processed. It implies that the number of jobs in the buffer increases to b_{opt} at r'_i for some i and no job is being processed at this time instant. Let us denote by t the starting time of the job processed first after r'_i. If $t > r'_{i+1}$, the number of jobs in the buffer becomes at least $b_{\text{opt}}+1$ at r'_{i+1}, which contradicts the assumption. Next, if $t = r'_{i+1}$, the number of jobs that arrive at r'_{i+1} should be one because otherwise, the number of jobs in the buffer exceeds $b_{\text{opt}}+1$ even if one job is started at r'_{i+1}. Even if it is satisfied, the number of jobs in the buffer at r'_{i+1} becomes $b_{\text{opt}}+1-1=b_{\text{opt}}$ in $\mathscr{S}$, although a job is being processed. Therefore, it also contradicts the assumption. Moreover, if $r'_i < t < r'_{i+1}$, the number of jobs in the buffer can be decreased to $b_{\text{opt}}-1$ for (P) by starting that job at r'_i, and it contradicts the optimality of $\mathscr{S}$. To summarize, the objective value of the solution of (P') should be $b_{\text{opt}}+1$ and hence $\mathscr{S}$ can be converted to $\mathscr{S}'$.

Next, it will be shown that $\mathscr{S}'$ can be converted to $\mathscr{S}$. Let us construct a feasible solution of (P) with $b_i \equiv 1$ from $\mathscr{S}'$ by moving the jobs under processing from the buffer to the machine outside the buffer. To show that the objective value of this solution does not exceed b_{opt}, it suffices to show that the maximum number of jobs in the buffer does not exceed b_{opt} in $\mathscr{S}'$ when no job is being processed. Assume without loss of optimality that $\mathscr{S}'$ has no unforced idle time: no job can be started earlier without changing the processing order. Consider that in $\mathscr{S}'$, the number of jobs in the buffer is $b_{\text{opt}}+1$ at r'_i and no job is being processed at this time instant. If no job is started and finished until r'_{i+1}, the number of jobs in the buffer becomes at least $b_{\text{opt}}+2$ at r'_{i+1} in $\mathscr{S}'$ and it contradicts the assumption on the objective value. Therefore, at least one job should be started and finished after r'_i and before r'_{i+1}. However, in this case an idle time occurs from r'_i, which contradicts the assumption. Hence, the number of jobs in the buffer does not exceed b_{opt} in $\mathscr{S}'$ when no job is being processed.

4.2 Integer Programming Formulation (IP')

Based on (P') introduced in the preceding subsection, (P) with $b_i \equiv 1$ will be formulated as another integer programming problem (IP').

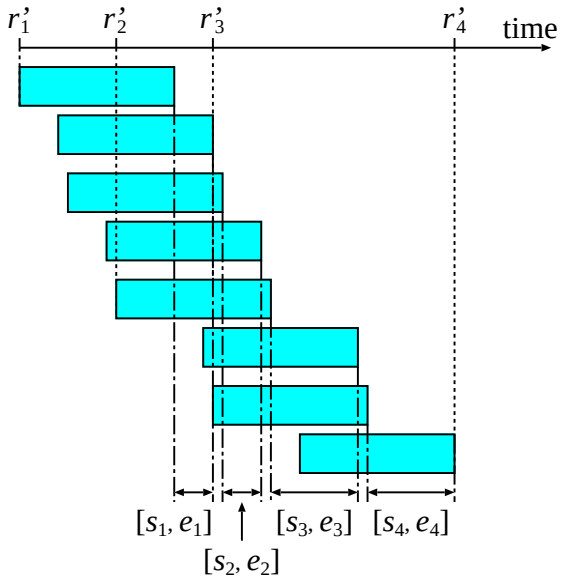


Fig. 5 Definition of W_i

Let us assume that the time horizon is divided into n_{R} intervals $[r'_1, r'_2)$, $[r'_2, r'_3]$, …, $[r'_{n_{\mathrm{R}}-1}, r'_{n_{\mathrm{R}}})$ and $[r'_{n_{\mathrm{R}}}, \infty)$. To denote in which intervals job i is processed, let us introduce the set W_i defined by

$$W_i = \left\{ (s,e) \,|\, s \in W_i^{\mathrm{S}},\ e = \min_{\substack{t \in W_i^{\mathrm{E}} \\ s \le t}} t \right\}, \tag{27}$$

$$W_i^{\mathrm{S}} = \{ r'_j + p_i \,|\, r_i \le r'_j < r'_{n_{\mathrm{R}}} \} \cup \{ r'_j + 1 \,|\, r_i \le r'_j - p_i + 1 < r'_{n_{\mathrm{R}}} \}, \tag{28}$$

$$W_i^{\mathrm{E}} = \{ r'_j + p_i - 1 \,|\, r_i \le r'_j - 1 \} \cup \{ r'_j \,|\, r_i \le r'_j - p_i \}. \tag{29}$$

Then, the intervals in which job i is processed are determined by $(s,e) \in W_i$ such that the completion time C_i of job i satisfies $C_i \in [s_i, e_i]$. Figure 5 gives an example. The intervals in which job i is processed are:

- $C_i \in [s_1, e_1]$
  $[r'_1, r'_2)$ and $[r'_2, r'_3)$,
- $C_i \in [s_2, e_2]$
  $[r'_1, r'_2)$, $[r'_2, r'_3)$ and $[r'_3, r'_4)$,
- $C_i \in [s_3, e_3]$
  $[r'_2, r'_3)$ and $[r'_3, r'_4)$,
- $C_i \in [s_4, e_4]$
  $[r'_3, r'_4)$.

Hence, $W_i = \{(s_1, e_1), (s_2, e_2), (s_3, e_3), (s_4, e_4), \ldots\}$ in this case. By using W_i, let us define the decision variables $y_i^{(s,e)}$ $(1 \le i \le n, (s,e) \in W_i)$ so that $y_i^{(s,e)} = 1$ if $C_i \in [s, e]$, and $y_i^{(s,e)} = 0$

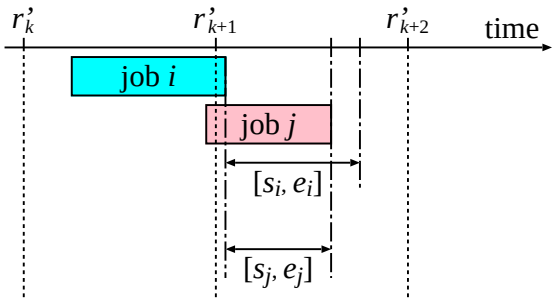


Fig. 6 A Feasible Solution of (IP') Corresponding to an Infeasible Schedule

otherwise. Let us also introduce $V_{ijk}^{(s,e)}$ $(1 \le i \le n, (s,e) \in W_i, 1 \le j < k \le n_{\mathrm{R}})$ defined by

$$V_{ijk}^{(s,e)} = \begin{cases} p_i, & r'_j \le s - p_i \wedge e \le r'_k, \\ \max\left\{\min(s, r'_k) - \max(e - p_i, r'_j), 0\right\}, & \text{otherwise.} \end{cases} \tag{30}$$

This $V_{ijk}^{(s,e)}$ denotes the minimum length of the processing time p_i of job i in the interval $[r'_j, r'_k)$, when job i is completed in the interval $[s,e]$. Furthermore, let us define the number of jobs arrived before r'_k $(1 \le k \le n_{\mathrm{R}})$ by $N_k = |\{j \,|\, r_j \le r'_k\}|$. Then, (P) with $b_i \equiv 1$ can be formulated as the following integer programming problem (IP'):

$$\min \ b_{\max}, \tag{31}$$

$$\text{s.t. } b_{\max} \ge N_k - 1 - \sum_{i=1}^{n} \sum_{\substack{(s,e)\in W_i \\ e \le r'_k}} y_i^{(s,e)}, \quad 1 \le k \le n_{\mathrm{R}}, \tag{32}$$

$$\sum_{(s,e)\in W_i} y_i^{(s,e)} \le 1, \quad 1 \le i \le n, \tag{33}$$

$$\sum_{i=1}^{n} \sum_{(s,e)\in W_i} V_{ijk}^{(s,e)} y_i^{(s,e)} \le r'_k - r'_j, \quad 1 \le j < k \le n_{\mathrm{R}}, \tag{34}$$

$$y_i^{(s,e)} \in \{0,1\}, \quad 1 \le i \le n, \ (s,e) \in W_i, \tag{35}$$

$$b_{\max} \in \mathbb{Z}. \tag{36}$$

Since (IP') formulates the problem to minimize the objective value of (P') minus 1, it is equivalent to (P) with $b_i \equiv 1$ from Theorem 2. The righthand side of (32) denotes the number of jobs in the buffer minus 1 at r'_k for (P'), and this equation corresponds to (21) in (IP). Equation (33) corresponds to (22), and it requires that job i is processed at most once until $r_n (= r'_{n_{\mathrm{R}}})$. Moreover, (34) corresponds to (23): it ensures that at most one job is processed at a time in the interval $[r'_j, r'_k)$.

It is obvious that a feasible schedule of (P') can be converted to a feasible solution of (IP') without changing the maximum buffer size. However, the reverse is not true: job overlaps may occur in a feasible solution of (IP'). For example, in the case of Fig. 6, $V_{ik,k+1}^{(s_i,e_i)} = V_{i,k+1,k+2}^{(s_i,e_i)} = V_{jk,k+1}^{(s_j,e_j)} = V_{j,k+1,k+2}^{(s_j,e_j)} = 1$ and thus (34) is satisfied, while the jobs overlap each other in both the processing orders $i \to j$ and $j \to i$. Nevertheless, a feasible schedule of (P') can be constructed from a feasible solution of (IP') without changing

the maximum buffer size. Suppose that $y_i^{(s_{k_i}, e_{k_i})} = 1$ holds for $(s_{k_i}, e_{k_i}) \in W_i$ and $1 \leq i \leq n$. Then, a feasible schedule is obtainable by sequencing the jobs in the nondecreasing order of $s_{k_i} - p_i$ without unforced idle times, where ties are broken by choosing the job with the shorter processing time. To verify that the objective value does not increase from $b_{\max} + 1$, it suffices to show that the completion time C_i of job i in the obtained schedule does not exceed $\min_{e_{k_i} \leq r'_j} r'_j$. Let job p be the first job that satisfies $C_q > r'_{j_E} = \min_{e_{k_q} \leq r'_j} r'_j$. Since jobs are sequenced without unforced idle times, there exists a block of jobs processed contiguously until job q. This block should start from r'_{j_S} for some j_S, and $r'_{j_S} \leq s_{k_i} - p_i$ holds for any job i in this block. Otherwise, such a job should be started before r'_{j_S}. Therefore, if $e_{k_i} \leq r'_{j_E}$ is also satisfied for any job i in the block, $V_{ij_S j_E}^{(s_{k_i}, e_{k_i})} = p_i$ and thus (34) should have been violated for $j = j_S$ and $k = j_E$. However, it contradicts the feasibility of the solution.

To show $e_{k_i} \leq r'_{j_E}$, assume that $r'_{j_E} < e_{k_i}$ holds for some $i(\neq q)$ in the block. Then, $s_{k_i} - p_i \leq s_{k_q} - p_q$ together with $e_{k_q} \leq r'_{j_E} < e_{k_i}$ yields $p_i > p_q$ and hence $s_{k_i} - p_i < s_{k_q} - p_q$ should be satisfied due to the tie-breaking rule in the construction procedure. There are two possibilities for s_{k_q} from (28): $s_{k_q} - p_q = r'_j - p_q + 1$ or $s_{k_q} - p_q = r'_j$ for some $j < j_E$. If $s_{k_q} - p_q = r'_j - p_q + 1$, $V_{ijj_E}^{(s_{k_i}, e_{k_i})} = r'_{j_E} - r'_j$ follows from $s_{k_i} - p_i < s_q - p_q \leq r'_j < r'_{j_E} < e_{k_i}$. Similarly, if $s_{k_q} - p_q = r'_j$, $s_{k_i} - p_i < s_{k_q} - p_q = r'_j < r'_{j_E} < e_{k_i}$ and thus $V_{ijj_E}^{(s_{k_i}, e_{k_i})} = r'_{j_E} - r'_j$ holds also in this case. These imply that no job other than job i can be processed between r'_j and r'_{j_E} from (34) for $j = j$ and $k = j_E$, which contradicts the assumption that $s_{k_q} = r'_j + 1$ or $s_{k_q} - p_q = r'_j$. Therefore, $e_{k_i} \leq r'_{j_E}$ is satisfied for any i in the block, and we can conclude that the above construction procedure generates a feasible schedule for (P') without increasing the maximum buffer size.

The primary reason why this type of formulation is not directly applicable to (P) is that it cannot distinguish whether a job is started just at r'_k or not. For example, suppose that job i satisfies $p_i < r'_{k+1} - r'_k$ for some k. In this case, the decision variables for job i do not distinguish $C_i = r'_k + p_i$ from $C_i = r'_k + p_i + 1, \ldots, r'_{k+1}$, although the occupied buffer space at r'_k becomes smaller only when $C_i = r'_k + p_i$.

The numbers of decision variables in (IP) and (IP') are $O(r_n n_R)$ and $O(n n_R)$, respectively, while the numbers of constraints in (IP) and (IP') (except (24), (25), (35) and (36)) are $(n + n_R + r_n - r_1 + 1)$ and $(n + n_R(n_R + 1)/2)$, respectively. It follows that the number of decision variables in (IP') is always smaller than that in (IP), but the number of constraints in (IP') is larger than that in (IP) when n_R is large and r_n is small. Since we cannot say that (IP') is always better than (IP) when $b_i \equiv 1$, the effectiveness of (IP') will be examined by numerical experiments.

5 Simple Heuristic Algorithm

To solve (P) by a general MIP solver takes a substantial computation time when the size of the problem becomes large. Therefore, in this section a simple algorithm will be proposed to solve (P) heuristically.

In this algorithm, job i with the smallest p_i/b_i is processed first. The details are given as follows.

(0) Let the set of unprocessed jobs be $\mathscr{U} := \{1, \ldots, n\}$ and the current time t be $t := 0$.
(1) Let k be $k := \arg\min_{r_i \leq t,\ i \in \mathscr{U}} p_i/b_i$. If $\{i \mid r_i \leq t,\ i \in \mathscr{U}\} = \emptyset$, let $t := \min_{i \in \mathscr{U}} r_i$ and go to (1).

(2) Let $t := t + p_k$ and the completion time C_k of job k be $C_k := t$.
(3) $\mathscr{U} := \mathscr{U} \setminus \{k\}$. If $\mathscr{U} = \emptyset$, terminate. Otherwise, go to (1).

6 Numerical Experiments

In this section, the effectiveness of the two formulations (IP) and (IP') as well as the heuristic algorithm in the preceding section will be examined by numerical experiments.

Problem instances were generated as follows. Processing times p_i and sizes b_i were generated from the integer uniform distributions in $[1, p_{\max}]$ and $[1, 10]$, respectively. Release dates r_i were generated from the integer uniform distribution in $[0, \lfloor \tau P \rfloor]$, where $P = \sum_{i=1}^{n} p_i$. Here, $p_{\max}$ and τ are the parameters to determine the maximum processing time and the maximum release date, respectively. These parameters were chosen as $p_{\max} \in \{10, 50, 100\}$ and $\tau \in \{0.5, 1.0, 1.5, 2.0\}$, and n, the number of jobs, was chosen as $n \in \{10, 20, \ldots, 50\}$. For each combination of n, $p_{\max}$ and τ, 10 instances were generated.

Gurobi Optimizer 5.02 and IBM ILOG CPLEX 12.5 were employed as general MIP solvers. Computation was performed on a desktop computer with a 3.3GHz Intel Core i7 3960X CPU and 32GB memory. The time limit was set to 3600 s.

6.1 Results for the Unit Size Instances

First, the unit size instances generated by setting $b_i \equiv 1$ will be considered. The results are summarized in Tables 1-3. In these tables, "var." denotes the average number of decision variables, "const." the average number of constraints, "Gurobi" the average CPU time in seconds by Gurobi Optimizer, and "CPLEX" the average CPU time in seconds by IBM ILOG CPLEX. In the case when all the 10 instances cannot be solved within 3600 s, the average CPU time was calculated over only optimally solved instances, and the number of optimally solved instances is shown in parentheses. From these tables, we can verify that the numbers of decision variables and constraints depend on $p_{\max}$ in (IP), while they do not in (IP'). Therefore, (IP) is better than (IP') when $p_{\max} = 10$, but (IP') becomes better as $p_{\max}$ increases. In addition, the average CPU time decreases for (IP') as τ increases, while no significant tendency is observed for (IP). As for the efficiency of the two MIP solvers, ILOG CPLEX seems to work better than Gurobi Optimizer.

6.2 Results for the General Instances

Next, the results for the general instances will be examined, which are summarized in Table 4. Please note that (IP') is unable to be employed in this case. From Table 4, the same tendency that the instances with a smaller τ is harder to solve is observed as for the unit size instances. It is also observed that Gurobi Optimizer works more efficiently than IBM ILOG CPLEX for the hard instances with $\tau = 0.5$, 1.0 and $n \geq 40$.

6.3 Effectiveness of Simple Heuristics

Finally, the effectiveness of the heuristic algorithm will be examined. Table 5 shows the average ratio of the objective value by the heuristic algorithm over the optimal (best if the

Table 1 Computational Results for Unit Size Instances ($p_{\max} = 10$)

n	τ	(IP)				(IP')			
		var.	const.	Gurobi	CPLEX	var.	const.	Gurobi	CPLEX
10	0.5	127.4	42.6	0.0	0.0	40.7	49.3	0.0	0.0
	1.0	268.3	67.9	0.0	0.0	58.5	56.0	0.0	0.0
	1.5	367.2	95.1	0.0	0.0	61.5	57.2	0.0	0.0
	2.0	470.4	110.2	0.0	0.0	67.3	59.5	0.0	0.0
20	0.5	556.3	92.2	0.1	0.0	214.8	166.8	0.1	0.1
	1.0	969.8	137.5	0.1	0.0	277.8	201.7	0.1	0.1
	1.5	1417.6	182.7	0.1	0.1	290.6	211.9	0.1	0.1
	2.0	2142.7	240.9	0.2	0.1	314.3	216.0	0.1	0.0
30	0.5	1271.8	135.9	0.2	0.1	533.2	357.7	0.5	0.3
	1.0	2316.3	211.2	0.3	0.1	674.9	428.9	0.7	0.4
	1.5	3714.1	295.6	0.5	0.1	709.0	439.6	0.3	0.2
	2.0	4619.3	358.8	0.5	0.1	702.8	442.1	0.2	0.2
40	0.5	2250.3	186.8	0.3	0.1	1006.5	657.1	2.5	1.4
	1.0	4480.8	293.7	0.6	0.3	1193.8	759.3	2.1	1.3
	1.5	6311.0	393.2	0.8	0.3	1273.0	778.8	0.9	0.8
	2.0	8180.4	494.8	1.0	0.3	1270.6	790.9	0.7	0.6
50	0.5	3398.3	222.5	0.5	0.4	1516.9	898.9	4.6	2.6
	1.0	6899.9	370.1	1.2	0.7	1872.1	1114.8	5.7	3.3
	1.5	9647.0	486.7	1.2	0.6	2000.7	1205.3	2.9	1.9
	2.0	13393.2	630.7	1.6	0.7	2073.5	1219.7	1.8	1.3

Table 2 Computational Results for Unit Size Instances ($p_{\max} = 50$)

n	τ	(IP)				(IP')			
		var.	const.	Gurobi	CPLEX	var.	const.	Gurobi	CPLEX
10	0.5	467.0	111.8	0.0	0.0	47.5	59.6	0.0	0.0
	1.0	1279.8	260.8	0.3	0.1	68.3	61.5	0.0	0.0
	1.5	1751.9	330.1	0.4	0.1	69.4	61.4	0.0	0.0
	2.0	2128.0	441.0	0.5	0.1	73.1	62.4	0.0	0.0
20	0.5	2284.5	268.3	0.5	0.2	270.1	213.6	0.2	0.1
	1.0	4711.0	528.7	1.4	0.4	320.9	221.1	0.1	0.1
	1.5	7408.7	813.8	2.2	0.5	329.9	223.6	0.1	0.1
	2.0	9571.8	962.1	2.6	0.5	343.6	227.5	0.1	0.1
30	0.5	5300.4	420.5	1.7	1.0	686.3	467.9	0.9	0.8
	1.0	11295.7	779.3	4.9	1.5	737.8	468.5	0.6	0.6
	1.5	15609.5	1126.4	4.4	1.3	780.5	480.4	0.5	0.3
	2.0	20969.5	1486.7	6.1	1.4	796.8	492.6	0.3	0.2
40	0.5	10211.2	570.7	10.0	3.5	1266.6	796.6	9.5	3.4
	1.0	19194.1	978.4	25.4	9.1	1383.4	844.7	9.3	2.5
	1.5	28693.2	1492.6	8.1	2.7	1434.4	849.5	1.0	1.0
	2.0	37356.3	1976.2	9.9	2.8	1444.1	849.3	0.7	0.6
50	0.5	15434.6	724.5	*18.4 (9)*	*11.3 (9)*	2052.0	1246.7	30.7	20.6
	1.0	29551.4	1303.4	21.8	8.2	2197.3	1267.2	7.5	4.4
	1.5	47567.4	2002.2	12.5	5.3	2262.1	1297.2	2.4	2.2
	2.0	60517.3	2537.4	16.9	6.2	2294.3	1307.5	1.9	1.4

time limit was reached) objective value by the MIP solvers. As a whole, the heuristic algorithm works better for the unit size instances than for the general instances. The reason why the ratio is small for the instances with $\tau = 0.5$ is that the objective value is large for these instances and hence the difference between the two objective values does not affect greatly on the ratio compared to the other instances.

Table 3 Computational Results for Unit Size Instances ($p_{\max} = 100$)

n	τ	(IP)				(IP')			
		var.	const.	Gurobi	CPLEX	var.	const.	Gurobi	CPLEX
10	0.5	1033.9	229.3	0.3	0.1	49.9	58.0	0.0	0.0
	1.0	2268.4	444.6	1.8	0.3	69.5	62.3	0.0	0.0
	1.5	2752.4	578.7	2.7	0.4	69.7	62.2	0.0	0.0
	2.0	4676.5	920.9	4.5	0.7	70.8	61.4	0.0	0.0
20	0.5	4518.7	494.5	3.0	2.0	285.4	220.5	0.2	0.1
	1.0	8581.4	908.3	7.0	1.4	320.4	217.4	0.2	0.1
	1.5	12594.1	1394.2	10.1	1.5	327.3	221.1	0.1	0.1
	2.0	18384.4	1834.0	11.7	2.0	333.5	225.3	0.1	0.0
30	0.5	10472.1	750.8	14.9	4.6	706.9	470.7	1.5	1.2
	1.0	21622.4	1573.0	17.4	7.3	785.0	489.2	0.7	0.6
	1.5	33562.3	2226.3	24.5	6.8	794.6	489.3	0.4	0.3
	2.0	42322.9	2858.3	17.3	5.0	801.2	489.3	0.3	0.2
40	0.5	18933.9	1026.0	*34.5 (8)*	*16.2 (8)*	1335.0	832.1	9.4	11.9
	1.0	37318.0	2043.3	39.2	17.0	1392.0	832.9	3.2	2.1
	1.5	55775.9	2820.2	21.7	9.7	1446.1	841.4	0.9	0.9
	2.0	75308.4	3751.9	26.6	10.0	1475.7	853.7	0.7	0.6
50	0.5	29886.9	1290.0	151.7	31.2	2094.9	1260.8	32.4	9.7
	1.0	57705.5	2422.8	358.4	96.6	2244.0	1301.9	13.8	7.7
	1.5	90760.3	3841.0	42.9	22.2	2292.7	1312.2	2.9	2.4
	2.0	121987.8	4825.9	40.0	20.3	2316.2	1322.5	1.7	1.6

Table 4 Computational Results for General Instances

n	τ	$p_{\max} = 10$		$p_{\max} = 50$		$p_{\max} = 100$	
		Gurobi	CPLEX	Gurobi	CPLEX	Gurobi	CPLEX
10	0.5	0.0	0.0	0.0	0.0	0.3	0.1
	1.0	0.0	0.0	0.3	0.1	1.7	0.3
	1.5	0.0	0.0	0.5	0.1	2.8	0.4
	2.0	0.0	0.0	0.6	0.2	4.3	0.8
20	0.5	0.1	0.1	1.7	1.0	8.0	3.5
	1.0	0.2	0.1	1.7	0.7	11.7	3.6
	1.5	0.1	0.1	3.0	1.0	11.9	1.9
	2.0	0.2	0.1	2.8	0.8	12.0	2.6
30	0.5	1.8	3.9	12.3	7.4	357.1	215.3
	1.0	0.4	0.3	11.0	33.9	43.7	77.0
	1.5	0.6	0.3	6.8	3.3	19.7	12.9
	2.0	0.7	0.2	6.4	2.1	25.9	9.6
40	0.5	1.5	1.3	*32.7 (9)*	*438.3 (8)*	*502.6 (8)*	*38.4 (4)*
	1.0	1.3	1.1	81.2	*444.4 (9)*	85.7	*38.7 (8)*
	1.5	1.0	0.5	16.6	6.4	60.2	51.3
	2.0	1.5	0.6	14.9	5.8	38.6	21.1
50	0.5	8.6	54.8	*447.9 (8)*	*162.4 (5)*	*256.5 (6)*	*277.4 (2)*
	1.0	3.4	2.2	91.4	*138.9 (8)*	289.1	190.7
	1.5	1.8	1.2	26.9	8.8	80.2	61.8
	2.0	2.3	1.4	47.5	16.0	83.5	33.7

7 Conclusion

In this study, the single-machine scheduling problem with a finite capacity input buffer was considered and the problem to minimize the required buffer size was treated. First, this problem was shown to be NP-hard in the strong sense. Next, an integer programming formulation was given. Then, another formulation for the unit size problem was derived by

Table 5 Average Ratio of Objective Value of Heuristics over Optimum

n	τ	$p_{\max} = 10$		$p_{\max} = 50$		$p_{\max} = 100$	
		$b_i \equiv 1$	general	$b_i \equiv 1$	general	$b_i \equiv 1$	general
10	0.5	1.100	1.248	1.203	1.468	1.125	1.342
	1.0	1.183	1.240	1.083	1.312	1.100	1.308
	1.5	1.100	1.228	1.000	1.315	1.283	1.688
	2.0	1.000	1.526	1.100	1.216	1.250	1.426
20	0.5	1.000	1.129	1.041	1.125	1.042	1.236
	1.0	1.067	1.181	1.120	1.341	1.145	1.274
	1.5	1.033	1.315	1.083	1.302	1.150	1.247
	2.0	1.000	1.423	1.300	1.303	1.000	1.465
30	0.5	1.028	1.121	1.074	1.176	1.066	1.226
	1.0	1.045	1.353	1.110	1.290	1.092	1.353
	1.5	1.050	1.353	1.200	1.349	1.108	1.324
	2.0	1.033	1.191	1.133	1.533	1.150	1.321
40	0.5	1.021	1.101	1.052	1.105	1.008	1.116
	1.0	1.053	1.175	1.000	1.172	1.056	1.233
	1.5	1.033	1.183	1.133	1.410	1.112	1.369
	2.0	1.100	1.380	1.133	1.189	1.167	1.343
50	0.5	1.012	1.056	1.006	1.116	1.028	1.098
	1.0	1.054	1.212	1.101	1.185	1.135	1.373
	1.5	1.125	1.302	1.133	1.422	1.117	1.418
	2.0	1.000	1.258	1.083	1.346	1.217	1.241

introducing an equivalent problem. A simple heuristic algorithm was also proposed. Finally, their effectiveness was examined by numerical experiments.

Future research directions will be to consider other objective functions such as total completion time, total weighted tardiness, to construct efficient exact and heuristic algorithms, and so on.

References

1. W.M. Nawijn, Minimum loss scheduling problems, European Journal of Operational Research, **56**, 364/369 (1992)
2. W.M. Nawijn, W. Kern, and S. Baas, Scheduling jobs with release times on a machine with finite storage, European Journal of Operational Research, **78**, 120/127 (1994)
3. N.G. Hall, M.E. Posner, and C.N. Potts, Preemptive scheduling with finite capacity input buffer, Annals of Operations Research, **70**, 399/413 (1997).
4. N.G. Hall, M.E. Posner, and C.N. Potts, Scheduling with finite capacity output buffers, Operations Research, **46**, S84/S97 (1998).
5. N.G. Hall, M.E. Posner, and C.N. Potts, Scheduling with finite capacity input buffers, Operations Research, **46**, S154/S159 (1998).
6.
7. D. Briskorn, et al. Complexity of single machine scheduling subject to nonnegative inventory constraints, European Journal of Operational Research, **207**, 605/619 (2010).
8. N. Boysen, S. Bock, and M. Fliedner, Scheduling of inventory releasing jobs to satisfy time-varying demand: an analysis and complexity. Journal of Scheduling, available online. DOI:10.1007/s10951-012-0266-0.
9. D. Briskorn, F. Jaehn, and E. Pesch, Exact algorithms for inventory constrained scheduling on a single machine, Journal of Scheduling, **16**, 105/115 (2013).
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).

MISTA 2013

Transfers in the on-demand transportation: the DARPT
Dial-a-Ride Problem with transfers allowed

Samuel Deleplanque • Alain Quilliot

Abstract Today, the on-demand transportation is used for elderly and disabled people for short distances. Each user provides a specific demand: a particular ride from an origin to a destination with hard time constraints like time windows, maximum user ride time, maximum route duration limits and precedence. This paper deals with the resolution of these problems (Dial-a-Ride Problems – DARP), including the possibility of one transshipment from a transfer point by request. We propose an algorithm based on insertion techniques and constraints propagation.

1 Introduction

An important Operations Research model for the management of flexible reactive transportation system is the Dial-a-Ride Problems. In the DARP, people (or a combination of people and goods) can order a ride in which they define mobility demands, give a pick-up location, a delivery location, two time windows, an upper bound on the duration of the demand, and the load related to the demand. The majority of these problems uses are related to the elderly and disabled people, but the latest research in transportation (connected cars, autonomous cars, etc.) associated with technological advances (mobile communication, geo-localization…) could provide new services for the optimization problem. DARP are also related to a shared service transportation because several users could be in one vehicle at the same time. This type of problem is complex because it incorporates hard time constraints like time windows. The optimization consists of creating the route of a fleet of vehicles in order to satisfy all (or the most possible) the mobility demands emanating from people. To determine these routes, one has to find a balance between two opposite things: the quality of service, and minimization of the total cost. This work integrates the possibility to make transshipment between two vehicles in order to satisfy a request. This transshipment is done by a dynamic transfer point, meaning this point is computed at the same time that the demand is included in a vehicle planning.

Samuel Deleplanque
LIMOS, UMR CNRS 6158, Bat. ISIMA, BLAISE PASCAL University, France
E-mail: deleplan@isima.fr

Alain Quilliot
LIMOS, UMR CNRS 6158, Bat. ISIMA, BLAISE PASCAL University, France
E-mail: quilliot@isima.fr

DARP can be modeled in different ways. There exists a number of ILP based methods [8], but the problem complexity is too high to use, most of which are NP-Hard because it also generalizes the Traveling Salesman Problem with Time Windows (TSPTW) [2] Therefore, the problem must be handled through heuristic techniques. [2] is one of the most important works on the subject and uses the Tabu search to solve it. Other techniques work well like dynamic programming (e.g. [14] and [1]) or variable neighborhood searches (VNS) (e.g. [13] and [6]). Moreover, a basic feature of DARP is that it usually derives from a dynamic context. So, algorithms for static DARP should be designed in order to take into account the fact that they will have to be adapted to dynamic and reactive context, which means synchronization mechanisms, interactions between the users and the vehicles, and uncertainty about fore coming demands. [15], and [9] later, developed heuristics based on insertion techniques. This methods are the most used technique in dynamic context or in a real exploitation. These techniques are a good solution when the people's requests have to be taken into account in a short time.

In this paper, we consider a generic DARP model with time windows and a mix QoS/Economical-Cost performance criterion, and propose algorithms for this model which are based upon randomized insertion techniques and constraint propagation, and so, which will easily adapt themselves to dynamic contexts, where demand package has to be inserted into (or eventually removed from) current vehicle schedules, in a very short time, while taking into account some probabilistic knowledge about fore coming demand packages. But, the main contribution of the paper is to allow transfer in the DARP. These transshipments are made dynamically and can be located everywhere. Little has been published on this subject, only [10]-[11] studied the problem. They express the problem by DARPT. The location of their transshipments points are known before the resolution. The authors use Tabu research, minimizing the total distance traveled by the fleet of vehicle. The closest problem is the Pickup and Delivery Problem with Transfers (PDPT).

There exist some exact methods to solve the PDPT, like [7], where the transfer points are those shaped by origin and destination nodes. [3] and [12] used a Branch-and-cut algorithm. As stated previously, the exact methods are not a good solution for solving the problem because it can't be used in a reactive context. The approximate methods are able to solve the problem in time. [4] created several rules for selecting the vehicle, assuming a demand given. Other rules help to trace the routes. The VNS is also used for this problem, [16] tested it on big instances (almost 200 requests). [17] worked on a heuristic where each pick-up and delivery nodes could be a transfer point. Their solution is based on insertion techniques and they allow the transshipments if the demand could not be inserted; [18] included the same algorithm in a dynamic context.

This paper is organized in the following manner: The next section will propose a model of the classic DARP. Then, we will explain how to handle the temporal constraints with a heuristic solution based on insertion techniques using propagation constraints. After, we will continue with the introduction of the DARPT model, framework and our updated solution based on insertion and propagation techniques. In the last part of the paper, the computational results will show the efficiency of our heuristics and we will compare the two solutions on the same instances.

2 The DARP: model and framework

2.1 The general Dial-a-Ride Problem

A *Dial a Ride* problem instance is essentially defined by a vehicle fleet $\mathcal{VH}$, a *transit* network $\mathbf{G} = (V, E)$, which contains at least a specific node *Depot*, and a *Demand* set $\mathcal{D} = (\mathcal{D}_i, i \in I)$, any demand $\mathcal{D}_i$ being defined as a 6-uple $\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i)$, where $\mathbf{o_i} \in V$ is the *origin* node of the demand $\mathcal{D}_i$, $\mathbf{d_i} \in V$ is the *destination* node of the demand $\mathcal{D}_i$, $\mathbf{\Delta_i} \geq 0$ is an

upper bound (*transit bound*) on the duration of demand $\mathcal{D}_i$'s processing, $\mathcal{F}(\mathbf{o_i})$ is a time window related to the time $\mathcal{D}_i$ starts being processed, $\mathcal{F}(\mathbf{d_i})$ is a time window related to the time $\mathcal{D}_i$ ends being processed, and $\mathbf{Q_i}$ is a description of the load related to $\mathcal{D}_i$.

Dealing with such an instance means planning the handling demands of $\mathcal{D}$, by the fleet $\mathcal{VH}$, while taking into account the constraints which derive from the technical characteristics of the network G, of the vehicle fleet $\mathcal{VH}$, and of the 6-uples $\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i)$, and while optimizing some performance criteria which is usually a mix of an economical cost (point of view of the fleet manager) and of QoS criteria (point of view of the users). This very general problem may be specialized according to several ways, depending on the structure of the fleet $\mathcal{VH}$ and on the way this fleet is allowed to answer various demands of $\mathcal{D}$.

The fleet $\mathcal{VH}$ may be heterogeneous, or, conversely, homogenous. In the first case, part of the problem consists in affecting the demands to the different classes of vehicles. The loads Q_i, $i \in I$, may be described as a set of objects, all endowed with their own characteristics (weight, volume, autonomous mobility…), or, conversely, summarized by a number (we say that they are *nominal*), which identifies any demand as a volume or as a weight of identical objects which are required to be transported from node o_i to node d_i. In the first case, part of the problem consists in identifying which combinations of objects may be simultaneously transported by a given vehicle. Also, in such a case, one may have to take into account the fact that loading and unloading processes are not neutral, and that their duration is likely to depend on the current load of the involved vehicles.

Load pre-emption (split loads) may be allowed, which means that a given load Q_i, $i \in I$, may be split into several blocks, every one being transported from node o_i to node d_i while using distinct vehicles, or while using twice a same vehicle. *Vehicle pre-emption* (**transfers**/transhipments) may be allowed too, which means that a given load Q_i, $i \in I$, may be transported in several steps, each step involving a specific vehicle: for instance, if we think into Q_i as into a single traveller, it may try to go from node o_i to node d_i while first using a shuttle and next a bus.

Temporal constraints related to the time windows $\mathcal{F}(o_i)$, $\mathcal{F}(d_i)$, $i \in I$, and to the transit bounds Δ_i, $i \in I$, may be more and less tight. The problem may have to be handled according to a *dynamic* context, (demands are not known in advance in an accurate way and must be processed "on line"): in such a case, one must take into account the way the system is supervised and the way its various components communicate with the users. Conversely, it may be set in a static context: all data are known in advance; the planning is computed and is run by the system. In this case, eventual divergences between the data which were used during the planning phases, and the situation the system has to really face, put what is called *robustness* at stake.

Finally, additional constraints may have to be tackled, such that cumulative constraints involving human, technical or financial mutualized renewable or non-renewable resources. In such a case, one may thing into linking the reduced problem with the RCPSP (*Resource Constrained Project Scheduling Problem*) framework.

Throughout this work, we deal with **homogeneous fleets** and with **nominal demands**, and therefore limit ourselves to static points of view. Still, we do not intend to restrict ourselves to *Standard Dial a Ride*: so, we pay special attention to cases when temporal constraints are tight, and handle the case when the transfers are allowed.

2.2 The Standard Case Framework

The general notations of sequences and algorithms are described as follows. In any algorithmic description, we use the symbol ← in order to denote the value assignment operator: $x \leftarrow \alpha$, means that the variable x receives the value α. Thus, we only use symbol = as a comparator. For any sequence (or list) Γ whose elements belong to some set Z, we set First(Γ) the first element of Γ and Last(Γ) is the last element of Γ. For any z in Γ, Succ(Γ, z) gives the successor

of z in Γ and Pred(Γ, z) gives the predecessor of z in the route Γ. For any z, z' in Γ, we note $z <<_\Gamma z'$ if z is located before z' in Γ; $z <<_\Gamma^{=} z'$ if $z <<_\Gamma z'$ or z = z'.
We consider here that **no pre-emption is allowed** (i.e. no **transfers** and no split loads), and that **no additional cumulative constraint** has to be taken into account. In such a case, we do not need to consider the whole transit network G = (V, E), and that we may restrict ourselves to the nodes which are either the origin or the destination of some demand, while considering that any vehicle which visits two such nodes in a consecutive way does it according to a shortest path strategy. This leads us to consider the node set {*Depot*, o_i, d_i, i ∈ I} as made with pairwise distinct nodes, and provided with some distance function DIST, which to any pair x, y in {*Depot*, o_i, d_i, i ∈ I}, makes correspond the shortest path distance from x to y in the transit network G.
As a matter of fact, we also split the *Depot* node according to its arrival or departure status and to the various vehicles of the fleet $\mathcal{VH}$, and we consider the **input data of a *Standard Dial a Ride*** instance as defined by the set {1..K = Card($\mathcal{VH}$)} of the vehicles of the homogenous fleet $\mathcal{VH}$, the common capacity $\mathcal{CAP}$ of a vehicle in $\mathcal{VH}$, the node set X = {*DepotD(k), DepotA(k)*, k = 1..K} ∪ {o_i, d_i, i ∈ I} and the distance matrix DIST, whose meaning is that, for any x, y in X, DIST(x, y) is equal to the length, in the sense of the length function l, of a shortest path which connects x to y in G: we suppose that DIST, satisfies the *triangle inequality*. Also, the following characteristics, which, to any node x in X, make correspond:

- its status $\mathcal{Status}$(x): *Origin*, *Destination*, *DepotA*, *Depot D*; we set *Depot* = *DepotD* ∪ *Depot A*;
- its demand index: $\mathcal{Dem}$(x) = i if x = o_i or d_i, and $\mathcal{Dem}$(x) = 0 else;
- its vehicle index $\mathcal{VI}$: $\mathcal{VI}$(*DepotA*(k)) = $\mathcal{VI}$(*DepotD*(k)) = k and $\mathcal{VI}$(x) = *Undefined* for any other node x ∈ X;
- its load $\mathcal{CH}$(x): if $\mathcal{Status}$(x) ∈ *Depot* then $\mathcal{CH}$(x) = 0; if $\mathcal{Status}$(x) = *Origin* then $\mathcal{CH}$(x) = Q_i;
- its twin node $\mathcal{Twin}$(x): if x = *DepotA*(k) then $\mathcal{Twin}$(x) = *DepotD*(k) and conversely; if x = o_i then $\mathcal{Twin}$(x) = d_i and conversely;
- its time window $\mathcal{F}$(x): for any k = 1..K, $\mathcal{F}$(*DepotA*(k)) = [0, + ∞ [= $\mathcal{F}$(*DepotD*(k)). Also, we suppose that any $\mathcal{F}$(x), x ∈ X, is an interval, which may be written $\mathcal{F}$(x) = [$\mathcal{F}$.min(x), $\mathcal{F}$.max(x)];
- its transit bound Δ(x): if x = o_i or d_i, then Δ(x) = Δ_i, and Δ(x) = Δ else, where Δ is an upper bound which is imposed on the duration of any vehicle tour.

According to this construction, we understand that the system works as follows: vehicle k ∈ {1..K}, starts its journey from *DepotD*(k) at some time t(*DepotD*(k)) and ends it into *DepotA*(k) at some time t(*DepotA*(k)), after having taken in charge some subset $\mathcal{D}$(k) = {$\mathcal{D}_i$, i ∈ I(k)} of $\mathcal{D}$: that means that for any i in I(k), vehicle k arrived in o_i at time t(o_i) ∈ $\mathcal{F}$(o_i), loaded the whole load Q_i, and kept it until it arrived in d_i at time t(d_i) ∈ $\mathcal{F}$(o_i) and unloaded Q_i, in such a way that t(d_i) - t(o_i) ≤ Δ_i. Clearly, solving the Standard Dial a Ride instance related to those data (X, DIST, K, CAP) will mean computing the subsets $\mathcal{D}$(k) = {$\mathcal{D}_i$, i ∈ I(k)}, the routes followed by the vehicles and the time values t(x), x ∈ X, in such a way that both economic performance and quality of service be the highest possible.

Remark on the Service Durations* and the *Waiting Times

Many authors include what they call *service durations* in their models. That means that they suppose that loading and unloading processes related to the various nodes of X require some time amount δ(x), (*service time*) and, so, that they distinguish, for any node x ∈ X, time values t(x) (beginning of the *service*) and t(x) + d(x) (end of the *service*). By the same way, some authors suppose that the vehicles are always running at their maximal speed, and make a difference between the time t*(x), x ∈ X, when some vehicle arrives in x, and the time t(x) when this vehicle starts servicing the related demand (loading or unloading process). We do not do it. Taking into account service times, which tends to augment the size of the variables of

the model and to make it more complex it, has really sense only if we suppose that the service times $\delta(x)$ depend on the current state (its current load) of the vehicle at the time the loading or unloading process has to be launched. Making explicitly appear waiting times $t(x) - t^*(x)$ is really useful if we make appear the speed profile as a component of the performance criteria. In case none of the situation holds, the knowledge of the routes of the vehicles and of the time value $t(x)$, $x \in X$, is enough to check the validity of a given solution and to evaluate its performance, and then it turns out that ensuring the compatibility of the model with data which involve service times and waiting times $t(x) - t^*(x)$, $x \in X$, is only a matter of adapting the times windows $\mathcal{F}(x)$, the transit bounds $\Delta(x)$, $x \in X$, and the distance matrix DIST.

Tours, Time-Valid Tours, Charge-Valid Tours, Valid Tour

In order to provide an accurate description of the **output data** of our standard Dial a Ride instance (X, DIST, K, $\mathcal{CAP}$), we need to talk about *tours* and related *time value sets*. A *tour* Γ is a sequence of nodes of X, which is such that:

- $\mathcal{Status}(\text{First}(\Gamma)) = \mathcal{Status}(\text{End}(\Gamma)) = \textit{Depot}$;
- $\mathcal{VI}(\text{First}(\Gamma)) = \mathcal{VI}(\text{End}(\Gamma))$;
- For any node x in Γ, $x \neq \text{First}(\Gamma), \text{End}(\Gamma)$, $\mathcal{Status}(x) \notin \textit{Depot}$;
- No node $x \in X$ appears twice in Γ;
- For any node $x = o_i$ (resp. d_i) which appears in Γ, the node $\mathcal{Twin}(x)$ is also in Γ, and we have: $x <<_\Gamma \mathcal{Twin}(x)$ (resp. $\mathcal{Twin}(x) <<_\Gamma x$).

This tour Γ is said to be ***load-valid*** *iff*: for any x in Γ, $x \neq \text{First}(\Gamma)$, we have $\mathbf{\Sigma}_{y \backslash y <<_\Gamma x} \mathcal{CH}(y) \leq \mathcal{CAP}$. And, this tour Γ is said to be ***time-valid*** *iff* it is possible to associate, with any node x in Γ, some time value $t(x)$, in such a way that: **(E1)**

- for any x in Γ, $x \neq \text{Last}(\Gamma)$, $\mathbf{t(Succ(\Gamma, x)) \geq t(x) + DIST(x, Succ(\Gamma, x))}$;
- for any x in Γ, $\mathbf{|t(\mathcal{Twin}(x)) - t(x)| \leq \Delta(x)}$ and for any x in Γ, $\mathbf{t(x) \in \mathcal{F}(x)}$.

The tour Γ is said to be ***valid*** if it is both time valid and load valid. For any pair (Γ, t) defined by some time-valid tour Γ and by some valid related time value set t, we may set $\mathbf{\mathcal{Glob}(\Gamma, t) = t(End(\Gamma)) - t(First(\Gamma))}$: this quantity denotes the global duration of the tour Γ and $\mathbf{\mathcal{Ride}(\Gamma, t) = \Sigma_{x \backslash \mathcal{Status}(x) = Origin} |\mathcal{Twin}(x) - t(x)|}$: this other quantity may be viewed as a QoS criteria, and denotes the sum of the duration of the individual trips of the demanders which are taken in charge by tour Γ. If A, B are two multi-criteria coefficients, we may define the performance criteria $\text{Cost}_{A, B}(\Gamma, t)$ as follows: $\mathbf{Cost_{A, B}(\Gamma, t) = A.\mathcal{Glob}(\Gamma, t) + B.\mathcal{Ride}(\Gamma, t)}$.

The Standard dial-a-ride model.

So, let us suppose that we deduced from the data G = (V, E), $\mathcal{VH}$ = (K, $\mathcal{CAP}$), $\mathcal{D} = (\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i), i \in I)$, a 4-uple (X, DIST, K, $\mathcal{CAP}$), and that we are also provided with 2 multi-criteria coefficients A and $B \geq 0$. Then we see that solving the related *Standard Dial a Ride Problem* instance means computing: for any vehicle index k in 1..K, a valid tour T(k) and a time value set $t = \{t(x), x \in X\}$ in such a way that: the restriction of t to any T(k), k = 1..K, defines a valid time value set related to T(k), the tour set $T = \{T(k), k = 1..K\}$ induces a partition of X, and the quantity $\mathcal{Perf}_{A, B}(T, t) \ \Sigma_{k = 1..K} \text{Cost}_{A, B}(T(k), t)$ is the smallest possible. In the operation research literature, there are several objective functions for the DARP. In the experiments section, we will use the one described above and another based on a mono-criterion related to the total distance.

3 Handling temporal constraints

Let Γ a tour. The algorithm which we are going to describe in section 4 will essentially be based upon the use of insertion techniques. Thus, we must be able to check in a fast way, whether the insertion of some demand $\mathcal{D}_i$ inside Γ will maintain the validity of Γ, and to get an

evaluation of the quality of this insertion. We are first going to provide ourselves with a package of constraint handling tools.

3.1 Testing the load-validity and the Time-validity

Checking the load-validity on Γ is easy. In order to be able to test the impact of the insertion of some demand into the tour Γ on the charge-validity of this tour, we associate, with any such a tour, the quantities $\mathcal{C}(\Gamma, x)$, $x \in \Gamma$, defined by: for any x in Γ, $\mathcal{C}(\Gamma, x) = \Sigma_{y \backslash y << = x} \mathcal{CH}(y)$. Then it comes that Γ is load-valid iff for any x in Γ, $\mathcal{C}(\Gamma, x) \leq \mathcal{CAP}$.
Checking the time validity of Γ, according to a current time window set $\mathcal{FS} = \{\mathcal{FS}(x) = [\mathcal{FS}.\min(x), \mathcal{FS}.\max(x)], x \in \Gamma\}$ may be performed through propagation of the following inference rules R_i, i = 1..5 performed by the **Propagate** procedure and we deduce the proposition 1 ([5]):

- **Rule R_1**: y = Succ(Γ, x); $\mathcal{FS}.\min(x) + DIST(x, y) > \mathcal{FS}.\min(y) \models \mathcal{FS}.\min(y) \leftarrow \mathcal{FS}.\min(x) + DIST(x, y)$; NFact ← y;
- **Rule R_2**: y = Succ(Γ, x); $\mathcal{FS}.\max(y) - DIST(x, y) < \mathcal{FS}.\max(x) \models \mathcal{FS}.\max(x) \leftarrow \mathcal{FS}.\max(y) - DIST(x, y)$; NFact ← x;
- **Rule R_3**: y = Twin(x); $x <<_\Gamma y$; $\mathcal{FS}.\min(x) < \mathcal{FS}.\min(y) - \Delta(x) \models \mathcal{FS}.\min(x) \leftarrow \mathcal{FS}.\min(y) - \Delta(x)$; NFact ← x;
- **Rule R_4**: y = Twin(x); $x <<_\Gamma y$; $\mathcal{FS}.\max(y) > \mathcal{FS}.\max(x) + \Delta(x) \models \mathcal{FS}.\max(y) \leftarrow \mathcal{FS}.\max(x) + \Delta(x)$; NFact ← y;
- **Rule R_5**: $x \in \Gamma$; $\mathcal{FS}.\min(x) > \mathcal{FS}.\max(x) \models \mathcal{Fail}$.

Procedure Propagate
Input : (Γ: Tour, L: List of nodes, $\mathcal{FS}$: Time windows set related to the node set of Γ):
Output : ($\mathcal{Res}$: Boolean, $\mathcal{FR}$: Time windows set related to node set of Γ);
Continue ← true;
While L ≠Nil and Continue do
z ← First(L); L ← Tail(L);
For i = 1..5 do Compute all the pairs (x, y) which make possible an application of the rule R_i and which are such that x = z or y = z;
For any such pair (x, y) do
Apply the rule R_i;
If NFact is not in L then Insert NFact in L;
If $\mathcal{Fail}$ then Continue ← false;
Propagate ← (Continue, $\mathcal{FS}$);

Proposition 1.
The tour Γ is time-valid according to the input time window set $\mathcal{FS}$ if and only if the $\mathcal{Res}$ component of the result of a call Propagate($\mathcal{FS}$, Γ) is equal to 1. In such a case, any valid time value set t related to Γ and $\mathcal{FS}$ is such that: for any x in Γ, $t(x) \in \mathcal{FS}(x)$.

Proof. The part (only if) of the above equivalence is trivial, as well as the second part of the statement. As for the part (if), we only need to check that if we set, for any x in Γ: $\mathcal{FS}(x) = [\mathcal{FS}.\min(x), \mathcal{FS}.\max(x)]$ and $t(x) = \mathcal{FS}.\min(x)$; then we get a time value set t ={t(x), $x \in X(\Gamma)$}which is compatible with Γ and $\mathcal{FS}$. ***End-Proof*.**

We denote by $\mathcal{FP}(\Gamma)$ the time window set which result from a call *Propagate*(Γ, Γ, $\mathcal{F}$),). $\mathcal{FP}(\Gamma)$ may be considered as the largest (in the inclusion sense) time window set which is included

into $\mathcal{F}$ and which is stable under the rules R_i, i = 1..5, and is called the *window reduction* of $\mathcal{F}$ through Γ.

3.2 Evaluating a tour

Let us consider now the tour Γ, provided with the window reduction set $\mathcal{FP}(\Gamma)$. We want to get some fast estimation of the best possible value $Cost_{A, B}(\Gamma, t) = A.\mathcal{Glob}(\Gamma, t) + B.\mathcal{Ride}(\Gamma, t)$, $t \in \mathcal{Valid}(\Gamma)$. We already noticed that it could be done through linear programming or through general shortest path and circuit cancelling techniques. Still, since we want to perform this evaluation process in a fast way, we design two ad hoc procedures EVAL1 and EVAL2. The EVAL1 procedure works in a greedy way, by first assigning to the node First(Γ) its largest possible time value, and by next performing a **Bellman process** in order to assign to every node x in Γ its smallest possible time value. The EVAL2 procedure starts from a solution produced by EVAL1, and improves it by performing a sequence of local moves, each move involving a single value t(x), x ∈ Γ. These procedures and the Proposition 2 are given below.

Procedure EVAL1. Input:(Γ: Tour); **Output:** (Val: Number, δ: value set)
For any x in Γ, let us set set: [a(x), b(x)] = $\mathcal{FP}(\Gamma)$;
δ(First(Γ)) ← b(First(Γ)); x ← First(Γ);
While x ≠ Last(Γ) do
    y < Succ(Γ, x); δ(y) ← Sup(a(y), δ(x) + DIST(x, y)) ;
    x ← y ; δ ← {δ(x), x ∈ Γ}; Val ← $Cost_{A, B, C}(\Gamma, \delta)$;
EVAL1 ← (Val, δ);

Procedure EVAL2. Input:(Γ: Tour); **Output:** (Val: Number, δ: value set)
For any x in Γ, let us set set: [a(x), b(x)] = $\mathcal{FP}(\Gamma)$;
For any x in Γ do δ(x) ← EVAL1(Γ, $\mathcal{FS}$).δ; Continue ← true;
While Continue do
  Search for x in Γ such that one of the two statements **(E2)** or **(E3)** below is true:
  - **(E2)**: $(\lambda_x < 0) \wedge (\mathcal{Status}(x) \in \{Origin, DepotD\}) \wedge (\delta(x) \neq Inf(b(x), \delta(Succ(\Gamma, x) – DIST(x, Succ(\Gamma, x)))$;
  - **(E3)**: $(\lambda_x > 0) \wedge (\mathcal{Status}(x) \in \{Destination, DepotA\}) \wedge (\delta(x) \neq Sup(a(x), \delta(Pred(\Gamma, x) + DIST(Succ(\Gamma, x)), x))$;

  If $\mathcal{Fail}$(Search) then Continue ← false;
  Else
    If **(E2)** then δ(x) ← Inf(b(x), δ(Succ(Γ, x) – DIST(x, Succ(Γ, x)));
    If **(E3)** then (δ(x) ← Sup(a(x), δ(Pred(Γ, x) + DIST(Succ(Γ, x)), x));
EVAL2 ← (Val = $Cost_{A, B}(\Gamma, \delta)$), δ);

Proposition 2.
Both EVAL1 and EVAL2 yield a time value set δ which is compatible with Γ and $\mathcal{F}$ (with Γ and $\mathcal{FP}(\Gamma)$). Besides, if B = 0, then EVAL1 yields an optimal value Val, that means yields the smallest possible value $Cost_{A, B}(\Gamma, \delta)$, $\delta \in \mathcal{Valid}(\Gamma, \mathcal{F})$.

Proof. As in the description of both procedures EVAL1 and EVAL2, we suppose that for any x in Γ, the time window $\mathcal{FP}(\Gamma)$ may also be written $\mathcal{FP}(\Gamma)$ = [a(x), b(x)]. The first part of the above statement is trivial. In case B = 0, minimizing $Cost_{A, B}(\Gamma, \delta)$ means minimizing δ(Last(Γ)) – δ(First(Γ)). We must deal with two cases:

- **First Case**: there exists x ∈ First(Γ) such that: δ(x) = a(x) and for any y such that x $<<_\Gamma^=$ y $<<_\Gamma$ Last(Γ), we have: δ(Succ(Γ, y)) – δ(y) = DIST(y, Succ(Γ, y));

Then the stability of $\mathcal{FP}(\Gamma)(x)$ under the inference rule $\mathbf{R_3}$ allows us to deduce $\delta(Last(\Gamma)) = a(Last(\Gamma))$, and the result since $\delta(First(\Gamma)) = b(First(\Gamma))$.

- **Second Case**: for any x in $X(\Gamma)$, $x \neq Last(\Gamma)$, we have $\delta(Succ(\Gamma, x)) - \delta(x) = DIST(x, Succ(\Gamma, x))$.

Then the result comes in an immediate way. ***End-Proof.***

Γ being some valid tour, we denote by VAL1(Γ) and VAL2(Γ) the values respectively produced by the application of EVAL1 and EVAL2 to Γ.

4 An insertion algorithm for tightly constrained instances of the standard DARP

4.1 The insertion mechanism

It works in a very natural way. Let Γ be some valid tour, let $\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i)$ be some demand whose origin and destination nodes are not in Γ, and let x, y be two nodes in Γ, such that $x <<_\Gamma^= y$. Then we denote by INSERT(Γ, x, y, i) the tour which is obtained by locating o_i between x and Succ(Γ, x) and locating d_i between y and Succ(Γ, y). We say that the tour INSERT(Γ, x, y, i) results from the *insertion of demand D_i into the tour Γ according to the insertion nodes x and y*. The tour INSERT(Γ, x, y, i) may not be valid. So, before anything else, we must detail the way the validity of this tour is likely to be tested.

Testing the Load-Admissibility of INSERT(Γ, x, y, i).
We only need to check with a procedure **Test-Load**, that for any z in Segment(Γ, x, y) = { z such that $x <<_\Gamma^= z <<_\Gamma^= y$} we have, $\mathcal{C}(\Gamma, z) + Q_i \leq \mathcal{CAP}$.

Testing the Time-Admissibility of INSERT(Γ, x, y, i).
It should be sufficient perform a call *Propagate*(Γ, $\{o_i, d_i\}$, $\mathcal{FP}(\Gamma)$), while using the list $\{o_i, d_i\}$ as a starting list. Still, such a call is likely to be time consuming. So, in order to make the testing process go faster, we introduce several intermediary tests, which aim at interrupting the testing process in case non-feasibility. The first test *Test-Node* aims at checking the feasibility of the insertion of a node u, related to some load Q, between two consecutive node z and z' of a given tour Γ. It only provides us with a necessary condition for the feasibility of this insertion:

Procedure Test-Node
Input: (Γ, z, z': nodes in Γ, u: node out Γ, Q: load); **Output:** Boolean
Let us set, for any x in Γ, $[a(x), b(x)] = \mathcal{FP}(\Gamma)(x)$; Let us set: $[\alpha, \beta] = \mathcal{F}(u)$;
Test node $\leftarrow (a(z) + DIST(z, u) \leq \beta) \wedge (\alpha + DIST(u, z') \leq b(z')) \wedge (a(z) + DIST(z, u) + DIST(u, z') \leq b(z')) \wedge (\mathcal{C}(\Gamma, z) + Q \leq \mathcal{CAP})$;

The second test *Test-Node1* (based on *Test-Node*) aims at checking the feasibility of the insertion of an origin/destination node u, v, related to some load Q, between two consecutive node z and z' of a given tour Γ. So, testing the admissibility of a tour INSERT(Γ, x, y, i) may be performed through the following procedure:

Procedure Test-Insert
Input: (Γ, x, y, i); **Output:** (Test: Boolean, Val: Number);
If $x \neq y$ then Test $\leftarrow$ *Test-Node*(Γ, x, Succ(Γ, x), o_i, Q_i) $\wedge$ *Test-Node*(Γ, y, Succ(Γ, y), d_i, Q_i)
Else Test $\leftarrow$ *Test-Node1*(Γ, x, Succ(Γ, x), o_i, d_i, Q_i);
If Test = 1 then Test $\leftarrow$ *Test-Charge*(Γ, x, y, i);
If Test = 1 then (Test, $\mathcal{F1}$) $\leftarrow$ *Propagate*(Γ, $\{o_i, d_i\}$, $\mathcal{FP}(\Gamma)$);

If Test = 1 then Val ← EVAL1(INSERT(Γ, x, y, i), *F1*).Val else Val ← Undefined;
Test-Insert ← (Test, Val – Val1(Γ));

4.2 The insertion process

So, this process takes as input the demand set $\mathcal{D} = (\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i), i \in I)$, the 4-uple (X, DIST, K, *CAP*) which we defined in section 2, and 2 multi-criteria coefficients A and $B \geq 0$, and it works in a greedy way through successive insertions of the various demands $\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i)$ of the demand set $\mathcal{D}$. The basic point is that, since we are concerned with tightly constrained time windows and transit bounds, we use, while designing the INSERTION algorithm, several constraint propagations tricks. Namely, we make in such a way that, at any time we enter the main loop of this algorithm, we are provided with:

- the set $I_1 \subset I$ of the demands which have already been inserted into some tour T(k), k = 1..K;
- current tours T(k), k = 1..K: for any such a tour T(k), we know the related time windows $\mathcal{FP}(T(k))(x)$, $x \in T(k)$, as well as the load values $\mathcal{C}(T(k), x)$, $x \in T(k)$, and the values VAL1(T(k)) and VAL2(T(k));
- the knowledge, for any i in $J = (I - I_1)$ of the set FREE(i) of all the 4-uple (k, x, y, v), k = 1..K, x, y ∈ T(k), v ∈ $\mathcal{Q}$, such that a call **Test-Insert**(T(k), x, y, i) yields a result (1, v). We denote by N-FREE(i) the cardinality of the set V-FREE(i) = {k = 1..K, such that there exists a 4-uple (k, x, y, v) in FREE(i)}: N-FREE(i) provides us with the number of vehicles which are still able to deal with demand $\mathcal{D}_i$.

Then, the INSERTION algorithm works according to the following scheme (1-4):

1 – The process selects a demand i_0 in J, among those demands which are the most constrained, that means which are such that N-FREE(i_0) and Card(Free(i_0)) are small. **(E4)**

2 – Then, in a second step, it picks up (k_0, x_0, y_0, v_0) in FREE(i_0) which simultaneously corresponds to one of the smallest values v, and to one of the smallest values EVAL2(INSERT(T(k), x, y, i_0)).Val – VAL2(T(k)): more specifically it first builds the list *L-Candidate* of the N_1 (up to five) 4-uples (k, x, y, v) in FREE(i_0) with best (smallest value v). For any such a 4-uple, it computes the value w = EVAL2(INSERT(T(k), x, y, i_0)).Val – VAL2(T(k)), and it orders *L-Candidate* according to increasing values w. Then it randomly chooses (k_0, x_0, y_0, v_0) among those $N_2 \leq N_1$ first 4-uples in *L-Candidate*. N_1 and N_2 become two parameters of the INSERTION procedure. **(E5)**

3 - It inserts the demand $\mathcal{D}_{i0}$ into T(k_0) according to the insertion nodes x_0, y_0, which means that it replaces T(k_0) by INSERT(T(k_0), x_0, y_0, i_0). Then, it defines, for any i ∈ J, the set Λ(i) as being the set of all pairs (x, y) such that there exists some 4-uple (k_0, x', y', v) in FREE(i), which satisfies: **(E6)**

- (x' = x) or ((x' = x_0) and x' = Pred(T(k_0), x)) or ((x' = x_0 = y_0) and (x' = Pred(Pred(T(k_0), x))));
- (y' = y) or ((y' = y_0) and y' = Pred(T(k_0), y)) or ((y' = x_0 = y_0) and (y' = Pred(Pred(T(k_0), y))))

4 - Finally, it performs, for any pair (x, y) in Λ(i), a call **Test-Insert**(T(k_0), x, y, i), and it updates FREE(i) and N-FREE(i) consequently.

Procedure INSERTION

Input: (N_1 and N_2: Integer, *the demand set* $\mathcal{D} = (\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i), i \in I)$, *the 4-uple (X, DIST, K, CAP) defined above, and 2 multi-criteria coefficients A and* $B \geq 0$);

Output: (T: tour set, t: time value set, *Perf*: induced $Perf_{A, B}(T, t)$ value, *Reject*: rejected demand set);

For any k = 1..K do

T(k) ← {*DepotD(k), DepotA(k)*}; t(*DepotD*(k)) = t(*DepotA*(k)) ← 0;

$I_1 \leftarrow$ Nil ; J $\leftarrow$ I ; $\mathcal{Reject} \leftarrow$ Nil;
For any i $\in$ J do
FREE(i) $\leftarrow$ all the possible 4-uple (k, x, y, v), k = 1..K, x, y $\in$ {*DepotD(k), DepotA(k)*}, x $<<_{T(k)}^{=}$ y, v = EVAL2({*DepotD(k),* o_i, d_i, *DepotA(k)*}).Val; N-FREE(i) $\leftarrow$ K;
While J $\neq$ Nil do
Pick up some demand i_0 in J as in **(E4)**; Remove i_0 from J;
If FREE(i_0) = Nil then $\mathcal{Reject} \leftarrow \mathcal{Reject} \cup \{i_0\}$
Else
Derive from FREE(i_0) the $\mathcal{L}$-*Candidate* list and Pick up (k_0, x_0, y_0, v_0) in $\mathcal{L}$-*Candidate* as in **(E5)**;
T(k_0) $\leftarrow$ INSERT(T(k_0), x_0, y_0, i_0); $\delta \leftarrow$ EVAL2(T(k_0)).δ; Insert i_0 into I_1 ;
For any x in T(k_0) do t(x) $\leftarrow \delta$(x);
For any i $\in$ J do
Λ(i) $\leftarrow$ {all pairs (x, y) such that there exists some 4-uple (k_0, x', y', v) in FREE(i), which satisfies **(E6)**
For any pair (x, y) in Λ(i) do
(Test, Val) $\leftarrow$ **Test-Insert**(T(k_0), x, y, i);
Remove (k_0, x, y, v) from FREE(i) in case such a 4-uple exists and update N-FREE(i) consequently;
If Test = 1 then insert (k_0, x, y, Val) into FREE(i) and update N-FREE(i) consequently;
$\mathcal{Perf} \leftarrow \mathcal{Perf}_{A, B}$(T, t);
INSERTION $\leftarrow$ (T, t, *Perf*, *Reject*);

Since the above instruction may be written in a non-deterministic way, the whole INSERTION algorithm becomes non-deterministic inside some **MONTE-CARLO framework**. This process keeps the best result (the pair (T, t) such that |*Reject*| is the smallest possible, and which is such that, among those pairs which minimize |*Reject*|, it yields the best $\mathcal{Perf}_{A, B}$(T, t) value).

5 Dial a ride problem with transfers

We are going to deal now with the case when *transfers* are allowed, that means then the load Q_i related to some demand $\mathcal{D}_i$ = (o_i, d_i, Δ_i, $\mathcal{F}(o_i)$, $\mathcal{F}(d_i)$, Q_i), may be handled in several successive steps, each step involving some vehicle k $\in$ K, which make the charge Q_i go from some origin or relay node x to some relay or destination node y. Transfers means here that, at any time during the transportation, while the load Q_i is always handled as a whole, the route it follows may be split into several sub-route, all those sub-routes being taken in charge by distinct vehicles. As a matter of fact, we are going to restrict here ourselves to the case when no more than 2 vehicles are allowed to perform such a transportation task: though this restriction is not going to induce any true restriction on concepts and methods, it will help us in describing them; also, **practical applications are such that they will hardly allow a same demand to be handled by more than 2 or 3 different vehicles**.

5.1 The insertion mechanism

The basic point is here that, in case a given load Q_i has to be successively handled by two vehicles k and k', the set X, which arises from a construction process involving only origin/destination nodes, is likely not to be sufficient to describe the route of the vehicles: the two related routes T(k), T(k') will have to intersect and exchange load Q_i in a relay node z, which will be neither an origin node o_j nor a destination node d_j. Those relay nodes, which may be nodes inside a large scale transit network, are not known in advance: so, we feel that

we should try to handle those exchange nodes in an implicit way and to deal with them in a dynamic way.

Extending the node set X into an implicit node set Z.
In order to put this in a formal way, we first need to extend X in order to make appear the relay nodes. Since we want to handle those relay node in a dynamic way, we suppose that X may be embedded into some (eventually infinite) implicit node set Z such that $X \subset Z$: Z may be a large scale, eventually infinite, set. For any pair of nodes z, z' in Z, we suppose that we are able to compute some distance $\mathcal{Dist}(z, z')$, in such a way that for any x, x' in X, Dist(x, x') = DIST(x, x'). Depending on the way Z is defined, this can be done in several ways: for instance, if nodes of X are related to some point in the 2-dimensional affine plane R^2, and if DIST represents the Euclidian distance, then Z may be the whole 2-dimensional plane R^2, and $\mathcal{Dist}$ may denote the Euclidian distance, which we are able to compute in an effective way once we know the 2D-coordinates of both points z and z'.
It comes that the input of the *Dial a Ride problem with transfers (**DARPT**)* is going to be defined, as in the standard version, by a transit network G = (V, E), a vehicle fleet $\mathcal{VH}$ = (K, $\mathcal{CAP}$), a demand set $\mathcal{D} = (\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i), i \in I)$, by a 4-uple (X, DIST, K, $\mathcal{CAP}$), and with 2 multi-criteria coefficients A and $B \geq 0$, augmented with some node set Z, which will derive from the transit network G and which will be such that $X \subset Z$, and by a function $\mathcal{Dist}$, with domain Z.Z, which will be such that: for any x, y in X, DIST(x, y) = $\mathcal{Dist}(x, y)$. Also, since we are going to handle the node set Z a dynamic way, we should be able to create new active nodes from existing ones. So, we suppose that we are provided with a function $\mathcal{Midst}$, which, from any pair of nodes z, z' in Z, compute a new node z" = $\mathcal{Midst}(z, z')$ in Z, in such a way that Dist(z, z") and Dist(z", z) are no larger than some fraction λ.Dist(z, z'), with $\lambda < 1$.

Building Relay Nodes: the Implicit Set Z*.
Additional nodes in Z – X are going to be used as relay nodes. Any such an active relay node will appear in 2 tours, once as an emitter node and once as a receiver node. Since we would like to keep on with the kind of model which we have been using for the standard version of the *Dial a Ride* problem, we also would like to make in such a way that all nodes which are going to appear in a tour family T be distinct. In order to do it, we define the implicit node set Z* as follows:

- $Z^* = X \cup \{(z, i, -1), (z, i, 1), i \in I, z \in Z\}$: the meaning of node (z, i, -1) is that if such a node becomes active, then it will appear as emitter node for load Q_i inside some tour T(k), which means that load Q_i is first going transported from o_i to z by vehicle k, and next from z to d_i by some other vehicle k', $k \neq k'$. It comes that node (z, i, 1) will appear in a symmetric way in tour T(k').
- for any node z in Z, we set: $\mathcal{Node}(z, i, -1) = \mathcal{Node}(z, i, +1) = z$;
- for any node x in X we set: $\mathcal{Node}(x) = x$.

We extend the $\mathcal{Status}$, $\mathcal{Twin}$, $\mathcal{Dem}$, $\mathcal{CH}$, Δ, and $\mathcal{F}$ functions by setting, for every $z \in Z$, $i \in I$:

- $\mathcal{Status}(z, i, -1) = \textit{Out-Reload}$, $\mathcal{Status}(z, i, +1) = \textit{In-Reload}$;
- $\mathcal{Twin}(z, i, +1) = \{(z, i, -1)$, $\mathcal{Twin}(z, i, -1) = \{(z, i, +1)$;
- $\mathcal{Dem}(z, i, +1) = \mathcal{Dem}(z, i, -1) = i$;
- $\mathcal{CH}(z, i, -1) = -Q_i$, $\mathcal{Node}(z, i, +1) = Q_i$;
- $\Delta(z, i, -1) = \Delta(z, i, +1) = +\infty$;
- $\mathcal{F}(z, i, -1) = \mathcal{F}(z, i, +1) = [0, +\infty[$.

Clearly, the distance function $\mathcal{Dist}$ may be also extended in a canonical way to Z*.Z*.

Tours, Valid Tours, Tour Family, Covering Tour Family, **Active Nodes**.
A tour, in the sense of the *DARPT*, becomes a sequence Γ of nodes of Z*, which is such that: $\mathcal{Status}(\text{Start}(\Gamma)) = \textit{DepotD}$, $\mathcal{Status}(\text{End}(\Gamma)) = \textit{DepotA}$, $\mathcal{VI}(\text{Start}(\Gamma)) = \mathcal{VI}(\text{End}(\Gamma))$, for any node x

in Γ: x ≠ Start(Γ), x ≠ End(Γ), $\mathcal{S}tatus$(x) ∉ *Depot*, no node x ∈ Z* appears twice in Γ, and for any demand i ∈ I, one of the configuration below occurs, which excludes the others:

- o_i and d_i appear in Γ, $o_i <<_\Gamma d_i$, and no node (z, i, ε), ε ∈ {-1, 1} is in Γ;
- none among o_i, d_i , (z, i, ε), ε ∈ {-1, 1}, appears in Γ;
- o_i and (z, i, -1) are in Γ, such that $o_i <<_\Gamma$ (z, i, -1), and none among d_i, (z, i, + 1), is in Γ;
- d_i and (z, i, +1) are in Γ, such that (z, i, +1) $<<_\Gamma d_i$, and none among o_i, (z, i, - 1), is in Γ;

As stated in the classic DARP section, we say that: such a tour Γ is ***load-valid*** iff : for any x in Γ, x ≠ Start(Γ), we have $\Sigma_{y \backslash y <<_\Gamma x} \mathcal{H}(y) \leq \mathcal{CAP}$ and such a tour Γ is ***time-valid*** iff it is possible to associate, with any node x in Γ, some time value δ(x) ≥ 0, in such a way that:

- for any x in Γ, x ≠ Last(Γ), **δ(Succ(Γ, x)) ≥ δ(x) + $\mathcal{D}ist$(x, Succ(Γ, x))**;
- for any x in Γ \ Status(x) ∉ {*Out-Reload*, *In-Reload*}, **$|\delta(\mathcal{T}win(x)) - \delta(x)| \leq \Delta(z)$**;
- for any x in Γ, δ(x) ∈ $\mathcal{F}$(x).

In case δ exists, it is called a *valid time value set* related to Γ. In case the tour Γ is both time-valid and charge-valid, we say that it is ***valid***. Clearly, a feasible solution of our *Dial a Ride with transfers* problem cannot be defined as a family T = {T(k), k = 1..K} of pairwise disjoint valid tours. We must link tours which involve *Out-Reload* and *In-reload* nodes related to a same demand. In order to do it, we consider some subset J of the *Demand Index* set I, some tour collection T = {T(1)..T(K)}, and we say that T defines a ***covering collection*** for J if the tours T(1)..T(K) are pair-wise disjoint valid tours, their union contains the whole set {o_i, d_i, i ∈ J} (they contain no node x such that $\mathcal{D}em$(x) ∈ I – J), and every tour T(k), k = 1..K, starts with the node *DepotD*(k) and ends with the node *DepotA*(k). The nodes (z, i, ε), z ∈ Z, i ∈ J, ε ∈ {-1, 1} which appear in $\cup_{k = 1..K}$ T(k) define the *active relay node set* of the tour collection T. We denote it by ACT(T). The tour collection T = {T(k), k = 1..K} is a *valid covering collection* for J *iff* :

- for every k = 1..K, the tour T(k) is load-valid;
- the collection T = {T(k), k = 1..K} is a ***covering collection*** for J
- there exists some time value set t = {t(x), x ∈ $\cup_{k = 1..K}$ T(k)} such that: for any k = 1..K, the restriction of t to the nodes of T(k) defines a valid time value set related to T(k); and, for every active *Out-Reload* node x in ACT(T), we have t($\mathcal{T}win$(x)) ≥ t(x);

In case such a time value set t exists, we say that T is *time-valid* and t is called a *valid time value set* related to T.

The Dial a Ride Problem with Transfers model.

Input: *A transit network G = (V, E), a vehicle fleet $\mathcal{VH}$ = (K, $\mathcal{CAP}$), a demand set $\mathcal{D}$ = ($\mathcal{D}_i$ = (o_i, d_i, Δ_i, $\mathcal{F}(o_i)$, $\mathcal{F}(d_i)$, Q_i), i ∈ I), by a 4-uple (X, DIST, K, $\mathcal{CAP}$), and by 2 multi-criteria coefficients A and B ≥ 0, augmented with the node set Z, such that X ⊂ Z, and with the function $\mathcal{D}ist$, with domain Z.Z, such that: for any x, y in X, DIST(x, y) = $\mathcal{D}ist$(x, y).*

Output: *A valid covering collection T = {T(k), k = 1..K} for the Demand Index set I, a related valid time value set = {t(x), x ∈ ACT(T)}, such that the quantity $\mathcal{P}erf_{A, B}$(T, t) is the smallest possible.*

5.2 Handling the relay nodes: an insertion mechanism

Once again, we want to deal with the above model by successively inserting the demands $\mathcal{D}_i$, i ∈ I, into a tour collection T, until this tour collection defines a valid covering collection for I. In order to do it, we first need to explain which kind of insertion mechanisms we intend to use. We are going to use two insertion operators: INSERT and INSERT2.

The **INSERT** operator works as in section 4.1: k being some vehicle index, x, y being two nodes in T(k) such that $x <<^{=}_{T(k)} y$, i being some demand index which is such that neither o_i nor

d_i is in T(k), INSERT(T(k), x, y, i) denotes the tour which is obtained through insertion of o_i between x and Succ(T(k), x) in T(k) and by insertion of d_i between y and Succ(T(k), y) in T(k). The **INSERT2** operator works by inserting demand $\mathcal{D}_i$ into two distinct tours: k, k' being two distinct vehicle indices, x, y being two nodes in T(k) such that $x <<^{=}_{T(k)} y$, x', y' being two nodes in T(k') such that $x' <<^{=}_{T(k')} y'$, z being some relay node in Z, INSERT2(i, k, k', x, y, x', y' , z) denotes a pair of tours (INSERT2(i, k, k', x, y, x', y' , z).First, INSERT2(i, k, k', x, y, x', y' , z).Second) in such a way that:

- INSERT2(i, k, k', x, y, x', y', z).First is the tour which is obtained through insertion of o_i between x and Succ(T(k), x) in T(k) and by insertion of (z, i, -1) between y and Succ(T(k), y) in T(k);
- INSERT2(i, k, k', x, y, x', y', z).Second is the tour which is obtained through insertion of (z, i, 1) between x' and Succ(T(k'), x') in T(k') and by insertion of d_i between y' and Succ(T(k), y') in T(k').

5.3 A general insertion scheme

We notice that the two operators which we described above have quite different impacts on the way a global insertion schema is going to work. While testing the feasibility of an application of the INSERT operator is a local task, which only involve dealing with the T(k) tour, testing the feasibility of the INSERT2 operator is likely to involve more than the T(k), T(k') tours. By the same way, handling the FREE(i) is going to become more complicated once we start introducing relay nodes and linking constraints. For this reason, we decompose the resolution process into two steps (1-2):
1 - We only use the INSERT operator, while proceeding as in the INSERTION procedure of section 4.2. Since we would like to use transfers and the related INSERT2 operator in order to make the whole tour system more efficient, we perform this first step while using stronger transit bounds Δ_i, $i \in I$, that means while making in such a way the riding times of the demanders get improved;
2 – The first step is likely to yield rejected demands, because of the stronger transit. So, the second step deals with those rejected demands while only using the INSERT2 operator.

That means that the whole process may be summarized as follows:

Dial-a-Ride with Transfers Insertion Algorithmic Scheme:
Continue ← true;
While Continue do
Compute updated transit bounds Δ^*_i, $i \in I$, such that for any $i \in I$, $\Delta^*_i \leq \Delta_i$; **(E7)**
1 - Apply the INSERTION(N_1, N_2) procedure of section 4.2, while replacing, for any $i \in I$, Δ_i by Δ^*_i;
Let *Reject* the resulting rejected demand index set, T the resulting valid covering collection associated with $I_1 = I - Reject$, and t the related valid time value set;
2 -: we keep on replacing, for any $i \in I$, Δ_i by Δ^*_i;
J ← *Reject*; *Reject* ← Nil; Initialize the sets FREE2(i), $i \in J$;
While $J \neq$ Nil do
Picks up some demand i_0 in J; Remove i_0 from J;
If FREE2(i_0) = Nil then *Reject* ← *Reject* $\cup \{i_0\}$;
Else
Derive from FREE2(i_0) a *Weak-L-Candidate* list of 7-uples; **(E8)**
Derive from *Weak-L-Candidate* a *L-Candidate* list, made with those 7-uples which are such that the replacement of ($T(k_1)$, $T(k_2)$) by INSERT2(i_0, k_1, k_2, x_1, y_1, x_2, y_2 , z) is going to maintain the validity of T; **(E9)**
If *L-Candidate* = Nil then *Reject* ← *Reject* $\cup \{i_0\}$

Else
Picks up (k_1, k_2, x_1, y_1, x_2, y_2, z) in *L-Candidate*;
Create two new active nodes related to (z, i_0, -1) and (z, i_0,1);
(T(k_1), T(k_2)) ← INSERT2(i_0, k_1, k_2, x_1, y_1, x_2, y_2 , z);
Update t; Update the time windows $\mathcal{F}(x)$, $x \in \cup\ k = {}_{1..K}\ T(k)$; **(E10)**
Update the sets FREE2(i), $i \in J$; Insert i_0 into I_1;
Update Continue;
Keep the best result (T, t, *Reject*, $Perf_{A, B, C}(T, t)$) which was obtained during this process.

Update-Δ imposes stronger transit bounds, creating this way a need for transfers in order to achieve better $Perf_{A, B}(T, t)$ values, we specify **(E7)** with this procedure as follows:

Function Update-Δ. Input: (λ : Number in Q, $\lambda > 1$); **Output:** (Δ^*_i, $i \in I$);
For i in I do : If $\Delta_i > \lambda$. DIST(o_i, d_i) then $\Delta^*_i \leftarrow \lambda$. DIST($o_i$, d_i) else $\Delta^*_i \leftarrow \Delta^*_i$;

5.4 The sets FREE2 and the construction of the weak-L-Candidate list

For any $i \in J$, the set FREE2-o(i) will be made with the pair (k, z) such that:

- the active node z is in T(k), different from Last(T(z)), [a(z), b(z)] denotes the time window $\mathcal{FP}(T)(z)$, z' gives Succ(T(k), z), and [α, β] denotes the time window $\mathcal{F}(o_i)$;
- $(a(z) + DIST(z, o_i) \le \beta) \wedge (\alpha + DIST(o_i, z') \le b(z')) \wedge (a(z) + DIST(z, o_i) + DIST(o_i, z') \le b(z')) \wedge (\mathcal{C}(\Gamma, z) + Q_i \le \mathcal{CAP})$.

In a same way, for any $i \in J$, the set FREE2-d(i) will be made with the pair (k, z) such that:

- $(a(z) + DIST(z, d_i) \le \beta) \wedge (\alpha + DIST(d_i, z') \le b(z')) \wedge (a(z) + DIST(z, d_i) + DIST(d_i, z') \le b(z')) \wedge (\mathcal{C}(\Gamma, z) + Q_i \le \mathcal{CAP})$.

So, for any (x_1, k_1) in FREE1-o(i_0), and for any (y_2, k_2) in FREE2-d(i_0), $k_1 \neq k_2$, we compute a relay node z through the following process summarized in 2 steps.
1 – First, let us recall that we are provided with a function *Midst*, which, from any pair of nodes z, z' in the node set Z, compute a new node z" = *Midst*(z, z') in Z, in such a way that *Dist*(z, z") and *Dist*(z", z) are no larger than some fraction λ.*Dist*(z, z'), with $\lambda < 1$;
2 - For any node y in T(k_1), we denote by *Close*(y, k_2) the first (in the sense of the relation $<<_{T(k2)}$) element x in T(k_1) such that $t(x) \ge t(y)$. Then, we apply the following *Exchange* function below.

Exchange function. **Input:** (x_1, k_1, y_2, k_2); **Output:** (y in T(k_1), x in T(k_2), z : relay node);
Compute y in T(k_1), such that :

- $x_1 <<_{T(k)} z$, *Close*(y, k_2) $<<^{=}_{T(k2)} y_2$, and DIST(y, *Close*(y, k_2) is the smallest possible ;

If y is undefined then ***Exchange*** ← *Undefined*
Else ***Exchange*** ← (y, Pred(*Close*(y, k_2)), *Midst*(z, U(z, l)));

The result which is produced by ***Exchange***(x_1, k_1, y_2, k_2) provides us with the parameters y_1, x_2, z, which would eventually allow us to perform the (T(k_1), T(k_2)) ← INSERT2(i_0, k_1, k_2, x_1, y_1, x_2, y_2 , z) instruction. So, the *Weak-L-Candidate* list **(E8)** will be defined by those 7-uple (k_1, k_2, x_1, y_1, x_2, y_2, z) which we obtain this way, and the *L-Candidate* list will defined by those among those 7-uple which are such the validity of the tour collection T will be preserved through application of the INSERT2 operator, ordered according to some auxiliary performance criteria.

5.5 Evaluate and testing the validity of an application of the INSERT2 operator

What remains to be done is explaining the way we check the validity of the tour collection T, in case we apply it some INSERT2 process. So, let us consider that we are provided with $I_1 \subset$

I, with a valid covering collection associated with I_1, with $i_0 \subset I - I_1$, with a 7-uple (k_1, k_2, x_1, y_1, x_2, y_2, z) as above, and that we intend trying to perform INSERT2(k_1, k_2, x_1, y_1, x_2, y_2, z). Clearly, checking the load validity of the two resulting tours $T(k_1)$ and $T(k_2)$ can be easily done through application of the following procedure:

Procedure Test-Load2; Input: (i_0, k_1, k_2, x_1, y_1, x_2, y_2, z); **Output**: Boolean;
Test-Load2 ← {For any z in Segment($T(k_1)$, x_1, y_1), $\mathcal{C}(\Gamma, z) + Q_{i0} \leq \mathcal{CAP}$} ∧ {For any z in Segment($T(k_2)$, x_2, y_2), $\mathcal{C}(\Gamma, z) + Q_{i0} \leq \mathcal{CAP}$}

As for the time validity, things become more difficult, since we need to take into account all the existing linking constraints, which means that the whole collection T is involved in the test. So, let us suppose that we just computed a copy Π of the tour collection which would result from the application of INSERT2(k_1, k_2, x_1, y_1, x_2, y_2, z), and that we just extend the DIST matrix in order to take into account the new active nodes (z, i_0, -1) and (z, i_0, 1). We may use the notation Succ and Pred for the whole collection Π as well as for a specific tour, since a given node in Z* is not going to appear more than once in Π. We denote by $\mathcal{FS}(x)$, $x \in ACT(\Pi) = \cup_{k = 1..K} \Pi(k)$ the current time windows related to the nodes x which appear in Π. What we do is to propagate the same 5 rules as in section 3.2 augmented with the following inference rules R_6 and R_7:

- Rule R_6: y = $\mathcal{Twin}(x)$; $\mathcal{Status}(x)$ = *Out-Reload*; $\mathcal{FS}.\min(x) > \mathcal{FS}.\min(y) \models \mathcal{FS}.\min(y) \leftarrow \mathcal{FS}.\min(x)$; NFact ← y;
- Rule R_7: y = $\mathcal{Twin}(x)$; $\mathcal{Status}(x)$ = *Out-Reload*; $\mathcal{FS}.\max(x) > \mathcal{FS}.\max(y) \models \mathcal{FS}.\max(x) \leftarrow \mathcal{FS}.\max(y)$; NFact ← x;

So, checking the time validity of Π, according to a current time window set $\mathcal{FS} = \{\mathcal{FS}(x) = [\mathcal{FS}.\min(x), \mathcal{FS}.\max(x)], x \in ACT(\Pi) = \cup_{k = 1..K} \Pi(k)\}$ is performed by application of a procedure *Propagate2* based on *Propagate*. We may consider, in case $\mathcal{Res}$ = 1, that the resulting time window set $\mathcal{FR} = \{\mathcal{FR}(x), x \in ACT(\Pi)\}$, is completely determined by Π and by the original time window set $\mathcal{F}$. So, we denote it by $\mathcal{FP}(\Pi)$, and we consider it as attached to any time-valid tour collection Π. Like in section 3, the tour collection Π is ***time valid*** iff the *Propogate2* function yields a positive $\mathcal{Res}$ signal.
As in Section 3, we need to evaluate the collection Π in case it is time-valid and compute some well-fitted related time value set δ. In order to do it, we only focus on the $\mathcal{Glob}$ component of the $\mathcal{Perf}$ criteria, and we apply a *Bellman* process: we start by providing every *DepotD* node x with a maximal $\mathcal{FP}(\Pi)(x).\max$ time value, and next, we do in such a way we assign any other node x with a time value δ(x) which is the smallest possible, taking into account the initial assignments δ(*DepotD*(k), k = 1..K, the current time windows $\mathcal{FP}(\Pi)(x)$, $x \in ACT(\Pi)$, the linking constraint and the distance constraints. Clearly, procedures *Propagate2* and *Evaluate2*, which will be at the core of the construction of the $\mathcal{L\text{-}Candidate}$ list in **(E8)**, are also being involved in **(E10)**, when it will come to updating the time value set t and the current time windows $\mathcal{F}$. Once the "($T(k_1)$, $T(k_2)$)) ← INSERT2(i_0, k_1, k_2, x_1, y_1, x_2, y_2 , z);" instruction have been performed, time windows $\mathcal{FP}(T)$ will be computed, as well as t = Evaluate2(T).δ.

5.6 Attempting an insertion with transfer and building the L-Candidate list.

So, let us suppose that we are provided with a non-inserted demand index i_0 and with some 7-uple (k_1, k_2, x_1, y_1, x_2, y_2, z). We want to try and, in case of success, to evaluate, an application of the process INSERT2 to (i_0, k_1, k_2, x_1, y_1, x_2, y_2, z). So, we perform the process **Try-Insert2**(i_0, k_1, k_2, x_1, y_1, x_2, y_2, z), while proceeding step by step:
First step: we perform a call *Test-Load2*(i_0, k_1, k_2, x_1, y_1, x_2, y_2, z);
Second step: we check that $DIST(o_{i0}, Succ(T(k_1), x_1) + Length(T(k_1), Succ(T(k), y_1) + \mathcal{Dist}(y_1, z) + \mathcal{Dist}(z, Succ(T(k_2), x_2) + Length(T(k_2), Succ(T(k_2), x_2), y_2) + DIST(y_2, d_{i0}) \leq \Delta_{i0}$;

Third step: we create temporarily two new active nodes z-out = (z, i_0, -1) and z-in = (z, i_0, -1) in Z*, and we augment the DIST matrix in such a way it will provide with the respective distances between z-out and z-in and their resepctive neighbours in $T(k_1)$ and $T(k_2)$.
Fourth step: we perform a call INSERT2(i_0, k_1, k_2, x_1, y_1, x_2, y_2, z) on Π; Time windows $\mathcal{FS}(x)$, in ACT(Π) are the current time windows $\mathcal{FR}(T)(x)$, if x is in ACT(T), and they are: $\mathcal{F}(o_{i0})$ if $x = o_{i0}$, $\mathcal{F}(d_{i0})$ if $x = d_{i0}$ and $[0, +\infty[$ if x = z-in or z-out;
Fifth step: we run the *Propagate2* procedure. In case of success, we run the *Evaluate2* procedure and get a resulting value $\mathcal{Val}$;
Sixth step: we restore the DIST matrix, the Π collection, and the ACT(Π) set.
At the end of this process, **Try-Insert2**(i_0, k_1, k_2, x_1, y_1, x_2, y_2, z) provides us with a 6-uple ($\mathcal{Res}$: Boolean, $\mathcal{Val}$: Number). The Boolean $\mathcal{Res}$ expresses the feasibility of an application of an INSERT2(i_0, k_1, k_2, x_1, y_1, x_2, y_2, z) call and the number $\mathcal{Val}$ provides us, in case $\mathcal{Res} = \mathcal{True}$, with an evaluation of such a call. We are now able to summarize the whole resolution process of the *DARPT*.

5.7 The whole process

The global process consists in a "for" loop, during which the parameter λ progressively decreases from an initial value Λ until 1: the length P of this loop is a parameter of the main procedure**.** Any iteration inside this loop works as described in section 5.3. Δ values are updated as in **(E7)**. A *first step* involves a call INSERTION(N_1, N_2) and yields some $\mathcal{Reject1}$ rejected demand index set, together with some pair (T, t), where T is a valid covering collection for I – $\mathcal{Reject1}$, and t is a related time value set. In case $\mathcal{Reject1}$ is not empty, a second step is performed, which involves a call to a procedure INSERTION2, which works while trying to insert the rejected demands through applications of the INSERT2 operator. The INSERTION2 procedure takes as input the 3-uple (T, t, $\mathcal{Reject1}$) which was computed through the *first step*, and proceeds, according to a "while" loop, in order to insert demands of $\mathcal{Reject1}$ through the INSERT2 operator. Any time it enters this main "while" loop, it is provided with a subset J of the $\mathcal{Reject1}$ set, which contains the demands which remain to be inserted and a current $\mathcal{Reject}$ set, a pair (T, t), related time windows $\mathcal{FP}(T)(x)$, $x \in ACT(T)$, and a related current active set ACT(T), and finally the sets FREE2-o(i) and FREE2-d(i), $i \in J$, defined as in VIII.4; We denote by N-FREE-o(i) (N-FREE-d(i)) the number of vehicle which appear in FREE2-o(i) (FREE2-d(i)). The process is composed of four main steps:

1. it picks up some demand i_0 in J: if there exists i such that N-FREE-o(i) = 1 or N-FREE-d(i) = 1, then i_0 is chosen in a random way among those demands in J which such that N-FREE-o(i) = 1 or N-FREE-d(i) = 1; else it is chosen in a random way among the demands in J which minimize the quantity N-FREE-o(i) = 1 + N-FREE-d(i); **(E11)**
2. It builds the $\mathcal{Weak\text{-}L\text{-}Candidate}$ list as in section 8.4, with those 7-uple (k_1, k_2, x_1, y_1, x_2, y_2, z) which are such that (k_1, x_1) ∈ FREE2-o(i_0), (k_2, x_2) ∈ FREE2-d(i_0), and (y_1, x_2, z) = *Exchange*(x_1, k_1, y_2, k_2); **(E12)**
3. For any 7-uple (k_1, k_2, x_1, y_1, x_2, y_2, z) in $\mathcal{Weak\text{-}L\text{-}Candidate}$, it tries the *Try-Insert2*(i_0, k_1, k_2, x_1, y_1, x_2, y_2, z) process, and, in case, the $\mathcal{Res}$ component of the result is true, it inserts (k_1, k_2, x_1, y_1, x_2, y_2, z) into a $\mathcal{L\text{-}Candidate}$ list, ordered according to increasing to the related $\mathcal{Val}$ component values; **(E13)**
4. In case $\mathcal{L\text{-}Candidate}$ is empty, then i_0 is inserted into $\mathcal{Reject}$, else:
   - the process picks up (k_1, k_2, x_1, y_1, x_2, y_2, z) in $\mathcal{L\text{-}Candidate}$: it proceeds through a random choice among the up to N_3 elements of $\mathcal{L\text{-}Candidate}$. N_3 becomes a parameter of INSERTION2; **(E14)**
   - It creates two new active nodes related to (z, i_0, -1) and (z, i_0,1) and it effectively performs the insertion: ($T(k_1)$, $T(k_2)$) ← INSERT2(i_0, k_1, k_2, x_1, y_1, x_2, y_2, z);

- It updates t, together with the time windows $\mathcal{FP}(x)$, $x \in ACT(T)$, through application of the *Propagate2* and *Evaluate2* procedures and the process updates the sets FREE2(i), $i \in J$;

Finally, the process keeps the best result (T, t, $\mathcal{Reject}$, $\mathcal{Perf}_{A, B}(T, t)$) which was ever obtained during this process. So the INSERTION2 and the DARPT-INSERTION procedure come may be described as follows:

Procedure INSERTION2(T1: partial tour collection, t1: related time value set, $\mathcal{Rej}$: Rejected Demand set): (T: tour set, t: time value set, $\mathcal{Perf}$: induced $\mathcal{Perf}(T, t)$ value, $\mathcal{Reject}$: rejected demand set, N_3: integers);
$J \leftarrow \mathcal{Rej}$; $\mathcal{Reject} \leftarrow$ Nil; $T \leftarrow T1$; $t \leftarrow t1$;
While $J \neq$ Nil do
Pick up some demand i_0 in J as in **(E11)**; Remove i_0 from J;
If FREE2-o(i_0) = Nil or FREE2-d(i_0) = Nil then $\mathcal{Reject} \leftarrow \mathcal{Reject} \cup \{i_0\}$
Else
Compute the $\mathcal{Weak\text{-}L\text{-}Candidate}$ list according to **(E12)**;
Build the $\mathcal{L\text{-}Candidate}$ list according to **(E13)**;
If $\mathcal{L\text{-}Candidate}$ = Nil then Reject ← Reject ∪ $\{i_0\}$
Else
Picks up (k_1, k_2, x_1, y_1, x_2, y_2, z) in $\mathcal{L\text{-}Candidate}$ according to **(E14)**;
Create two new active nodes related to (z, i_0, -1) and (z, i_0,1);
$(T(k_1), T(k_2)) \leftarrow$ INSERT2(i_0, k_1, k_2, x_1, y_1, x_2, y_2 , z);
Update t and $\mathcal{PF}(T)$ through application of the *Propagate2* and *Evaluate2* procedures;
For any $i \in J$, update FREE2-o(i) and FREE2-d(i);
Perf ← $\mathcal{Perf}(T, t)$;
INSERTION2 ← (T, t, Perf, Reject);

DARPT-INSERTION
Input: (N_1, N_2 **,** N_3, P: Integers, *A transit network G = (V, E), a vehicle fleet* $\mathcal{VH}$ *= (K,* $\mathcal{CAP}$*), a demand set* $\mathcal{D} = (\mathcal{D}_i = (o_i, d_i, \Delta_i, \mathcal{F}(o_i), \mathcal{F}(d_i), Q_i), i \in I)$*, by a 4-uple (X, DIST, K,* $\mathcal{CAP}$*), and by 2 multi-criteria coefficients A and B ≥ 0, augmented with the node set Z, such that X ⊂ Z, and with the function* $\mathcal{Dist}$*, with domain Z.Z, such that: for any x, y in X, DIST(x, y) =* $\mathcal{Dist}$*(x, y).*
Output: (T: Tour collection, t: time valued set, $\mathcal{Perf}$: Performance Value, $\mathcal{Reject}$: Rejected Demand Index set);
Δ-Aux ← Δ; Initialize λ with a large value Λ;
For p = 1..P do
$\Delta \leftarrow$ *Update-*Δ (λ, Δ); (T1, t1, $\mathcal{Perf1}$, *Reject1*) ← INSERTION1(N_1, N_2);
If $\mathcal{Reject1}$ = Nil then *DARPT-INSERTION* ← (T, t, $\mathcal{Perf}$, $\mathcal{Reject}$)
Else (T, t, $\mathcal{Perf}$, $\mathcal{Reject}$) ← INSERTION2(T1, t1, $\mathcal{Reject1}$, N_3);
$\Delta \leftarrow$ Δ-Aux ; Update λ: $\lambda \leftarrow \lambda - 1/P.(\Lambda - 1)$;
Keep the best result (T, t, $\mathcal{Reject}$, $\mathcal{Perf}_{A, B}(T, t)$) which was obtained during this process.

6 Computational experiments

All the insertion techniques based on the constraint propagation were implemented in C++ and each replication was run on the same thread of an Intel Q8300 (2.5 GHz).

6.1 Experiments on the DARP

Our first experiments deal with the randomly generated instances of [2]. To analyse the behaviour of our solution, we used the same objective function used in [19] and adapted in [20]. The instances have between 24 and 144 requests which have to be supported by a fleet of 3 to 13 vehicles. The maximum route duration is 480 for each vehicle and for each instance. The capacity is equal to 6 and the maximum ride time is 90. [19] used the objective function given in equation (4), the terms penalizing the violations have been removed. Thus, we minimize travel distance (c(s)), excess ride time (r(s), cf. (1)), passenger waiting (l(s), cf. (2)), the total duration $\mathcal{Glob}$ (g(s)) and early arrival (e(s), cf. (3)). We set the weight like in [19] and [20] to w_1=8, w_2=3, w_3=1, w_4=1, w_5= $|\mathcal{D}|$.

$$r(s) = \sum_{k=1}^{K} \sum_{i \in k} (Ride(i) - DIST(o_i, d_i)) \quad (1)$$

$$l(s) = \sum_{k=1}^{K} \sum_{x=succ(First(\Gamma_k))}^{pred(last(\Gamma_k))} Wait_{succ(x)}(C(\Gamma_k, succ(x)) - q_x) \quad (2)$$

$$e(s) = \sum_{k=1}^{K} \sum_{x=First(\Gamma_k)}^{pred(pred(last(\Gamma_k)))} F.Min(succ(x)) - (\delta(x) + DIST(x, succ(s))) \quad (3)$$

$$Cost = w_1 c(s) + w_2 r(s) + w_3 l(s) + w_4 g(s) + w_5 e(s) \quad (4)$$

The table 1 gives the values of the $\mathcal{COST}$ obtained with the proposed insertion techniques using constraint propagation. Gap is computed with the results of our insertion technique and the VNS such as *Gap←100.(VNS-TI)/TI*. We take best results over 25.10^4 replications with a variation in the values of N_1 and N_2 (each lower than 4). We noted only the objective function of the two works. So we compare our *Insertion Techniques* (IT) with the *Variable Neighbourhood Search* (VNS) and the *Genetic Algorithm* (GA). Refer to [19] and [20] for the other values. As with the VNS technique, we obtained results always better than the GA. Moreover, we often obtained better results than the variable neighbourhood search. So we found a large difference between [19] and the others works, but solutions obtained by us and [20] are close even though in R10a we obtain a large gap. In fact, time constraints of this instance are very tight and we use a simple learning algorithm without computing a precise order for introducing the demands already rejected.

Early arrivals have the largest weight in the objective function and our solution gives us numbers close to 0 (except for R10a) for the majority of the results. In this case, no vehicle arrives at a node before the beginning of a node's time window. Each large gap obtained (negative or positive) is related to the early arrivals. This is also why the solution based on a genetic algorithm [19] resulted in a very large cost. Our CPU times are close (or lower) to the VNS' runs with the same number of iterations (e.g. for the 25.10^4 replications we required less than one minute for the smallest instance (R1a) and 38 minutes for the biggest and hardest instance (R10a).

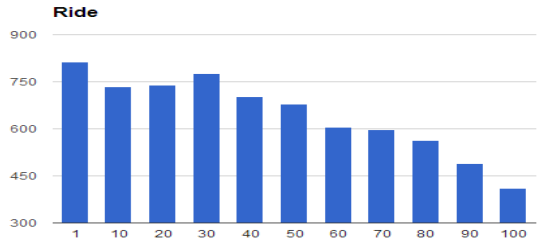
Table 1 - Insertion techniques (IT) compared to GA ([19]) and VNS ([20])

Inst.	|D|	GA [19]	VNS [20]	IT	Gap
R1a	24	4696	3234.6	3371.4	-4.1
R2a	48	19426	14640.2	9025.7	62.2
R3a	72	65306	15969.1	10780.8	48.1
R5a	120	213420	23852.0	14054.2	69.7
R9a	108	333283	13806.4	14175.7	-2.6
R10a	144	740890	25016.5	35359.5	-29.3
R1b	24	4762	2825.5	2927.6	-3.5
R2b	48	13580	5003.1	5066.5	-1.3

R5b	120	98111	12360.5	12528.9	-1.3
R6b	144	185169	16499.4	16339.4	1.0
R7b	36	9169	4601.7	4523.1	1.7
R9b	108	167709	13412.8	13564.9	-1.1
R10b	144	474758	16420.0	17546.5	-6.4
Average		179252.2	12895.5	12251.1	5.3

6.1 Experiments on the DARPT

Figure 1 - Indirect minimization of the Ride times



For the first short experimentation on the DARP with transfers, the **DARPT-INSERTION** algorithm was applied to the first instance of [2], the R1a, in order to analyse the evolution of the $\mathcal{Ride}$ time caused by the variation of the maximum ride time Δ. We report in Figure 1 eleven $\mathcal{Ride}$ times, each with 10 replications, where all the demands have been included in *valid* routes. These times are shorter and shorter (up to half of the first time) during all the executions, the $\mathcal{Glob}$ times had a small raise but not comparable to the $\mathcal{Ride}$ time. In a real context, so in a reactive context, depending on the time the system needs to accept or not the demand (virtual insertion and synchronization mechanisms included), this QoS criterion could be managed by the DARPT-INSERTION's value p.

For the second experimentation on the DARPT, we applied our solution to solve the DARPT on a set of randomly generated instances. Each instance is different by the size of the windows, the number of demand, and the number of cars. Like in [2], we randomly generated the coordinates of pick-up and drop-off nodes in the square of side 20. We split the square in 4 parts and the fleet $\mathcal{VH}$ in 4 sub-fleets $\mathcal{VH1}$, $\mathcal{VH2}$, $\mathcal{VH3}$, and $\mathcal{VH4}$ related to the sub-squares $\mathcal{EP1}$, $\mathcal{EP2}$, $\mathcal{EP3}$, and $\mathcal{EP4}$. $\mathcal{D}$ is classified in two sets: the transverse demands which are its origin node in a different sub-square than the destination and the local demands. For each instance studied here, 50% of the demands are local and uniformly set to the 4 sub-squares. We generated a different maximum user ride time which equal to the product of 20 and the distance between the origin node and the destination node. The capacity CAP equals 6 for each vehicle. Each demand has a large time windows (all the day, from 0 to 1440 minutes) and another tight (15 or 30 minutes), their $\mathcal{Status}$ is granted randomly. We performed 100 replications of 12 sets of 5 instances generated by the parameters written above. Table 3 gives the results. We provided $R1_c$ which is the rate of the demand inserted in the routes when the transfers are forbidden $R1_t$ is the same rate when transfers are allowed. We computed Gap such as $Gap \leftarrow (R1_t - R1_c)/(R1_c/100)$. All the results are average of the 100 replications. When comparing average rates obtained by each resolution, about 11.9% of demand can be inserted if the transfers are allowed in addition compared with no transfer. $R1_t$ and $R1_c$ are obviously better when the fleet has more cars (K=5), but if the gap is more important it means there are more possibility to do a transshipment. The first fourth instances have 3 times less demands inserted than the second set (with the same fleet). The R1t and R1c for the first set are a little less than 3 times less than

the second set. That is explain by the fact the second set are a bigger choice to included his demands.

Table 2. DARP classic Vs DARP with transfers

Inst.	|D|	K	Win. Size	RI_c	RI_t	Gap (%)	**Inst.**	|D|	K	Win. Size	RI_c	RI_t	Gap (%)
1	32	4	15	54.59	62.76	14.95	**7**	64	5	15	39.06	46.12	18.08
2	32	4	30	61.84	68.32	10.47	**8**	64	5	30	45.17	48.20	6.72
3	32	5	15	70.28	86.02	22.40	**9**	96	4	15	22.43	24.16	7.74
4	32	5	30	77.00	89.07	15.67	**10**	96	4	30	25.92	27.02	4.24
5	64	4	15	29.37	34.34	16.94	**11**	96	5	15	28.64	32.62	13.93
6	64	4	30	35.29	37.05	4.97	**12**	96	5	30	33.26	35.35	6.30

Acknowledgements We wish to thank you the *Conseil Regional d'Auvergne* and the FEDER of the European Union.

References

1. Chevrier, R., Canalda, P., Chatonnay, P., Josselin, D. (2006). Comparison of three algorithms for solving the convergent demand responsive transportation problem, ITSC'2006, 9th Int. IEEE Conf. on ITS, Toronto, Canada, 1096–1101.
2. Cordeau, J.-F., Laporte, G. (2003). A tabu search heuristic algorithm for the static multi-vehicle dial-a-ride problem, Transportation Research B 37, 579–594.
3. Cortes, C. E., Matamala, M., Contardo, C. (2010). The pickup and delivery problem with transfers, European Journal of Operational Research 200 p711-724 .
4. Cortes, C. E., Jayakrishnan, R. (2002). Design and operational concepts of high-coverage point-to-point transit system, Transportation Research Record 1783 p178-187.
5. Deleplanque, S., Quilliot, A. (2012). Insertion techniques and constraint propagation for the DARP. FedCSIS. 393–400. IEEE conference publications.
6. Healy, P., Moll, R. (1995). A new extension of local search applied to the dial-a-ride problem, European Journal of Operational Research 83, 83–104.
7. Kerivin, H., Lacroix, M., Mahjoub, A. R., Quilliot, A. (2008). The splittable pickup and delivery problem with reloads, European Journal of Industrial Engineering 2 - p112-133.
8. Laporte, G., Cordeau, J.F. (2007). The dial-a-ride problem: models and algorithms. Annals of Operations Research.
9. Madsen, O., Ravn, H., Rygaard, J. (1995). A heuristic algorithm for the DARP with time windows, multiple capacities, and multiple objectives, Annals of OR 60, 193–208.
10. Masson, R., Lehuédé, F., Péton, O. (2011). A tabu search algorithm for the Dial-a-Ride Problem with Transfers, Proceedings of the ICIESM.
11. Masson, R., Lehuédé, F., Péton, O. (2012). Simple Temporal Problems in Route Scheduling for the Dial–a–Ride Problem with Transfers, 2012, Lecture Notes in Computer Science, Volume 7298/2012, 275-291, DOI: 10.1007/978-3-642-29828-8_18.
12. Nakao, Y., Nagamochi, H. (2012). Worst case analysis for pickup and delivery problems with transfer, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E91-A (9) p2328-2334 .
13. Parragh, S.N., Doerner, K.F., Hartl, R.F. (2010). Variable neighborhood search for the dial-a-ride problem, Computers & Operations Research, 37, 1129–1138.
14. Psaraftis, H. (1983). An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. Transportation Science 17, 351–357.
15. Psaraftis, H., Wilson, N., Jaw, J., Odoni, A. (1986). A heuristic algorithm for the multi-vehicle many-to-many advance request DARP. Transportation Research B 20B, 243-257.

16. Masson, R., Lehuede, F., Peton, O. (2012). An adaptive large neighborhood search for the pickup and delivery problem with transfers, Transportation Science in press.
17. Shang, J. S., Cu, C. K. (1996). Multicriteria pickup and delivery problem with transfer opportunity. Computers & Industrial Engineering.
18. Thangiah, S., Fergany, A., Awam, S. (2007). Real-time split-delivery PDPTW with transfers, Central European Journal of Operations Research - 15 329-349.
19. R.M. Jorgensen, J. Larsen, and K.B. Bergvinsdottir Solving the DARP using genetic algorithms. Journal of the Operational Research Society, 58(10):1321-1331, 2007.
20. S.N. Parragh, K.F. Doerner, R.F. Hartl. Variable neighborhood search for the dial-a-ride problem. Computers & Operations Research, 37 p. 1129–1138, 2010.

MISTA 2013

Minimizing the number of late jobs on parallel machines with α time windows

Vincent Jeauneau · Philippe Chrétienne

Abstract We consider the problem of minimization the number of late jobs on parallel machines in presence of α time windows ($d_i = r_i + p_i + \alpha$ for all jobs). We prove the NP-completeness of this problem even when $m = 2$ and $d_i = d$ for all jobs. We develop a lower bound by solving a minimum cost flow problem associated to a relaxation of the problem. A set of heuristics are presented to minimize the number of late jobs based on a job-focused and machine-focused approach. A simulation experiment is realized to test the effectiveness of heuristics regarding the lower bound. This simulation has been computed for $1, 2, 5, 10, 15$ and 20 machines and it shows that the best heuristic has an average deviation to the bound of 4.88%, 6.35%, 7.38%, 7.5%, 7.87% and 7.64% respectively.

Keywords Parallel machines scheduling, number of late jobs, α time windows, lower bound

1 Introduction

Given a set of jobs $J = J_1, ..., J_n$ where each job has a release date r_i, a due date d_i, and a processing time p_i. Given a set of parallel identical machines $M = M_1, ..., M_m$. An important scheduling problem is to determine a schedule that minimizes the number of late jobs ($P|r_i|\sum U_i$). A late job is defined as a job which finishes its processing later than its due date. This criterion is important since, in many case, the cost penalty incurred by a late job depends by the fact that it is late and not how much it is. This problem is NP-hard even when all jobs have the same release date and the number of machines is two [2] or the due dates are common [7].

In this paper, we consider a special case of the general parallel machines scheduling problem in which all jobs satisfy $d_i = r_i + p_i + \alpha$ with α a positive constant of the problem.

Vincent Jeauneau
Thales Air Systems, Technical Directorate
voie Pierre-Gilles de Gennes, 91470 Limours, France
E-mail: vincent.jeauneau@thalesgroup.com

Philippe Chrétienne
LIP6, Université Pierre et Marie Curie
4 place Jussieu, 75252 Paris Cedex 05, France
E-mail: philippe.chretienne@lip6.fr

Since each job cannot delay its starting time by more than α, we call our problem the parallel machines scheduling problem with α time windows. Following the standard three-field notation proposed by Graham et al. [4], this problem is noted $P|d_i = r_i + p_i + \alpha|\sum U_i$. For the one-machine case, Jeauneau and Chrétienne [6] extend Garey et al. works [3] and develop a polynomial time algorithm to minimize the number of late jobs optimally.

The rest of the paper is organized as follow. In Section 2, we prove the NP-Completeness of the problem. Section 3, presents a method to obtain a lower bound. Section 4 are dedicated of the development of some heuristics. Experimental results are analyzed in Section 5. Finally, Section 6 summarizes the paper.

2 On the complexity of the parallel machines scheduling problem with α time windows

In this section, we show that the scheduling problem of minimizing the number of late jobs on parallel machines in which all jobs satisfy $d_i = r_i + p_i + \alpha$ is NP-hard. This result come from a reduction of the NP-Hard problem *PARTITION* [8] to our problem. Consider the *PARTITION* problem, in which we are given a finite set A of positive integers having an overall sum $2b$ and are asked whether there exists a subset of A that sums exactly to b ?

From an instance of *PARTITION*, we generate the following instance of the parallel machines scheduling problem with α time windows:

- $m = 2$,
- $\alpha = b$,
- J_i such as $r_i = 0$, $p_i = a_i$, $d_i = p_i + b$ for $i = 1, ..., n$,
- J_{n+1} such as $r_{n+1} = 0$, $p_{n+1} = b + 1$, $d_{n+1} = 2b + 1$,
- J_{n+2} such as $r_{n+2} = 0$, $p_{n+2} = b + 1$, $d_{n+2} = 2b + 1$.

if *PARTITION* has a solution $\left(A^{'}, A^{''}\right)$, then there exists a schedule where the jobs of $A^{'}$ are scheduled on the first machine and the jobs of $A^{''}$ are scheduled on the second machine in the interval $[0, b]$. Then, the two additional jobs $\{J_{n+1}, J_{n+2}\}$ are scheduled in the interval $[b, 2b + 1]$, one on the first machine, the other on the second (Fig. 1). If the parallel machines scheduling problem has a solution, then the jobs $\{J_{n+1}, J_{n+2}\}$ cannot be scheduled on the same machine. Thus, one is scheduled on the first machine and the other on the second machine. Moreover, both can be scheduled in the interval $[b, 2b + 1]$, therefore the jobs scheduled in the interval $[0, b]$ form a solution of *PARTITION* problem. We prove that the parallel machines scheduling problem with α time windows is NP-hard even when the number of machines is two and all jobs have the same release date. It should be noted that a symmetric approach (fix the d_i-values instead of the r_i-values) proves the NP-Completeness of this problem even when all jobs have the same due date.

3 Lower Bound

In section 2, we have proved the NP-Completeness of the problem $1|d_i = r_i + p_i + \alpha|\sum U_i$. We will present some heuristics to solve it in section 4. In order to compare the results obtained from heuristics, we develop a lower bound in this section. To compute this lower bound, we first relax the constraint of non-preemption. This relaxed problem is noted $P|pmtn; d_i = r_i + p_i + \alpha|\sum U_i$ and every optimal solutions of this problem are less or equal than optimal solutions of $P|d_i = r_i + p_i + \alpha|\sum U_i$.

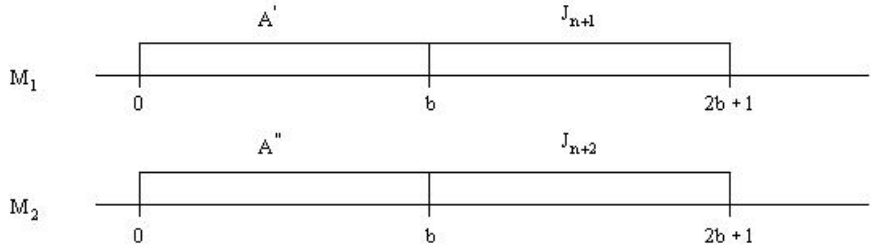


Fig. 1: Scheduling of jobs $\{J_1, ..., J_{n+2}\}$

Then we associate a minimum cost flow problem to the problem $P|pmtn; d_i = r_i + p_i + \alpha|\sum U_i$ such as solving the minimum cost flow problem leads to a lower bound for the problem $P|pmtn; d_i = r_i + p_i + \alpha|\sum U_i$ which is a lower for the problem $P|d_i = r_i + p_i + \alpha|\sum U_i$ too. Baker [1] solves the problem $P|pmtn; r_i|L_{\max}$ optimally with a similarly approach.

Let $J = \{J_1, ..., J_n\}$ be an instance of the problem $P|pmtr; d_i = r_i + p_i + \alpha|\sum U_i$ order as $d_1 \leq ... \leq d_n$. We define a minimum cost flow problem as follow:

- Let $t_1 < t_2 < ... < t_r$ be the ordered sequence of all different r_i-values and d_i-values.
- Let $I_K := [t_k, t_{k+1}]$ the intervals of length $l_k = t_{k+1} - t_k$ for $k = 1, ..., r-1$.
- A job vertex is associated to each job J_i and an interval vertex to each interval I_k.
- Let s a source and t a sink.
- There is an arc from s to each job vertex J_i of capacity p_i and cost 0.
- There is an arc from each interval vertex I_k to t of capacity $m \times l_k$ and cost 0.
- There exists an arc from J_i to I_k of capacity l_k and cost 0 iff job J_i can be processed in I_k (i.e $r_i \leq t_k$ and $t_{k+1} \leq d_i$).
- There is an arc from each job vertex J_i to t of capacity p_i and cost $1/p_i$.

Fig. 2 illustrate the graph $G(V, E)$ of the minimum cost flow problem defined above, where V is the set of vertices and E the set of arcs. Each arc is represented by a couple (x, y) where x is the capacity and y the unary cost of the arc.

A solution of maximum flow for such a graph $G(V, E)$ has a flow equal to $\sum_{i=1}^{n} p_i$. Indeed, for each job vertex J_i, there exists an arc (s, J_i) with a capacity p_i and another arc (J_i, t) with the same capacity. Since there only exits arcs from s to jobs vertices J_i, for $i = 1, 2, ..., n$, then we have a maximum flow solution in which the flow of arcs (s, J_i) and (J_i, t) are equal to p_i.

For each optimal solution of the minimum cost flow problem, we can build a schedule for the problem $P|pmtr; d_i = r_i + p_i + \alpha|\sum U_i$ as follow: for each interval vertex I_k, let x_{ik} denote the flow of arc (J_i, I_k), the job J_i executes a part of its processing time equal to x_{ik} in the interval of times $[t_k, t_{k+1}]$, for $i = 1, 2, ..., n$. Then, let θ_i denote the flow of arc (J_i, t), the job J_i executes, a part of its processing equal to θ_i, at the end of the schedule of an arbitrary machine, for $i = 1, 2, ..., n$.

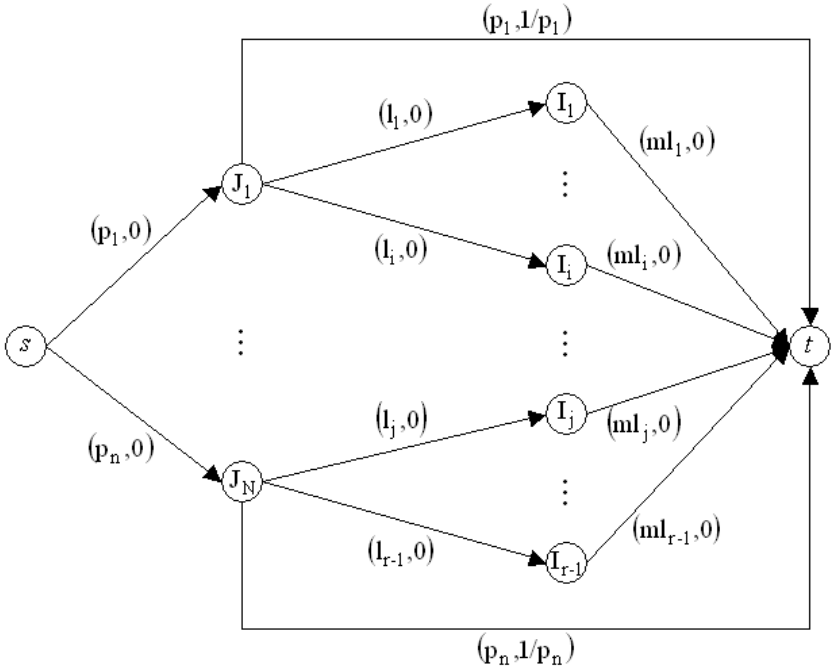


Fig. 2: Graph $G\,(V, E)$ of the minimum cost flow problem

The graph $G\,(V, E)$ is defined such as we have:

$$x_{ik} \leq l_j \text{ for } i = 1, 2, ..., n \text{ for } k = 1, 2, ..., r-1 \tag{1}$$

$$\sum_{i=1}^{n} x_{ik} \leq m \times l_k \text{ for } k = 1, 2, ..., r-1 \tag{2}$$

And from optimality of the minimum cost flow solution, we have:

$$\sum_{k=1}^{r_1} x_{ik} + \theta_i = p_i \quad \text{for } i = 1, 2, ..., n \tag{3}$$

From (1), we know that for each job vertex J_i and for each interval vertex I_k, the flow x_{ik} is less or equal than l_k. Since $l_k = t_{k+1} - t_k$, it implies that each job J_i cannot executes a part of its processing time greater than the length of the interval of time on which this part is executed. From (2), we know that for each interval vertex I_k we have $\sum_{i=1}^{n} x_{ik} \leq m \times l_k$. Therefore, the parts of processing times of all jobs J_i to be executed in $[t_k, t_{k+1}]$ have a total duration less or equal to $m \times l_k$. Since we dispose of m machines, these parts can be executed without overlapping whit another interval of time. From (3), we know that the flow of each job vertex J_i is exactly equal to p_i. Therefore, each job J_i are fully executed in the schedule.

We deduce that our schedule is a legal schedule for the problem $P|pmtn; d_i = r_i + p_i + \alpha| \sum U_i$ and we have the following property:

Property 1 An optimal solution of the minimum cost flow problem on the graph $G(V, E)$ is a lower bound of the problem $P|d_i = r_i + p_i + \alpha| \sum U_i$.

Proof Let Π an optimal schedule for an instance of the problem $P|d_i = r_i + p_i + \alpha| \sum U_i$. Let p_{ij} denote the j^{th} unity of p_i and c_{ij} its completion time in Π. We build a maximal flow f for the graph $G(V, E)$, associated to the problem instance, as follow:

- for each arc (s, J_i), there is a flow p_i;
- for each job J_i, if $c_{ij} \in [r_i, d_i]$ then increase by one the flow of arcs (J_i, I_k) and (I_k, t) such as $c_{ij} \in I_k$, otherwise increase by one the flow of (J_i, t).

The cost c of f is equal to $\sum_{i=1}^{n} y_i \times 1/p_i$, where y_i is the part of the job J_i which is not legally scheduled in Π, i.e $y_i = \sum_{j=1}^{p_i} p_{ij}$ such as $c_{ij} \notin [r_i, d_i]$. Then, for each job J_i, we have $y_i \leq p_i$ which imply $y_i \times 1/p_i \leq 1$. Moreover, when $U_i = 0$, the job J_i is not late scheduled, therefore $c_{ij} \in [r_i, d_i]$ for $j = 1, 2, ..., p_i$ and $y_i = 0$. Since $y_i \neq 0$ iff $U_i \neq 0$, we deduce that $\sum_{i=1}^{n} y_i \times 1/p_i \leq \sum_{i=1}^{n} U_i$.

From the definition of an optimal solution for the minimum cost flow problem, we have $c^* \leq c$ where c^* is the minimum cost of an optimal solution of maximum flow on the graph $G(V, E)$. We deduce that $c^* \leq \sum_{i=1}^{n} U_i$ and conclude that an optimal solution of the minimum cost flow problem on the graph $G(V, E)$ is a lower bound of the problem $P|d_i = r_i + p_i + \alpha| \sum U_i$.

Orlin [10] proposes a polynomial time algorithm for the minimum cost flow problem. This algorithm runs in $O(V \log V (E + V \log V))$. There is exactly n jobs vertices and at most $2n$ intervals vertices in $G(V, E)$, therefore the total number of vertices is in $O(n)$. There is n arcs between s and the jobs vertices, n arcs between jobs vertices and t, at most $2n$ between intervals vertices and t, and at most $n \times 2n$ arcs between jobs vertices and intervals vertices. Thus the total number of arcs is in $O\left(n^2\right)$. Therefore, the lower bound can be computed in $O\left(n^3 \log^2(n)\right)$.

We propose an amelioration of this lower bound by reducing the processing time of some jobs. Indeed, the value of the bound depends on the flow θ_i of each arc (J_i, t). The unary cost of these arcs is equal to $1/p_i$. Therefore, if we are able to reduce the processing time of a job J_i without change the flow θ_i, then the value of the lower bound will be increased.

We consider each interval vertex I_k, for k increasing for 1 to $r-1$. For each such interval vertex, we suppose that each job vertex J_i with an arc (J_i, I_k) has a flow exactly equal to l_k (this assumption is a relaxation on the capacity of arc (I_k, t)). Fixed these flows in $G(v, E)$ is equivalent to build the graph $G'\left(V', E'\right)$ where $V' = V - I_k$ and $E' = E - I_k^- - I_k^+$ with I_k^- and I_k^+ the inputs arcs and outputs arcs respectively. The capacity of arcs (s, T_i) and (T_i, t) for each job vertex J_i with an arc (J_i, I_k) in $G(V, E)$ is reduced by l_k in $G'\left(V', E'\right)$. For the graph $G'\left(V', E'\right)$, the cost of arcs (J_i, t) can be updated to the reverse of their capacities without violate the integrities constraints, i.e the flow of each job vertex J_i cannot increase the cost by more than one. Let c'^* and c^* denote the cost of the minimum cost flow solution for $G'\left(V', E'\right)$ and $G(V, E)$ respectively. If $c'^* > c^*$, then the quality of the solution for the graph $G'\left(V', E'\right)$ is better than the one for the graph $G(V, E)$ and the next intervals vertices $I_{k+1}, I_{k+2}, ..., I_{r-1}$ are considered on $G'\left(V', E'\right)$ otherwise continue on the procedure on graph $G(V, E)$.

Finally, the upper integer part $\lceil c^* \rceil$ can be used as a lower bound. Indeed, our aim is to minimize the number of late jobs. This objective is integer since $U_i = 1$ if J_i is late and $U_i = 0$ otherwise. Therefore the smallest integer which is greater than c^* define a lower bound, noted LB. This procedure solves a minimum cost flow problem for each interval

vertex I_k. There is no more than $2n$ intervals vertices, so this lower bound can be computed in $O\left(n^4 \log^2(n)\right)$.

It should be noted that this lower bound is independent of the structure of jobs and can be used for the general problem $P|r_i|\sum U_i$.

4 Heuristic approaches

In this section, we use two heuristic approaches to solve the m-machines scheduling problem with α time windows. These heuristic approaches are called job-focused and machine-focused. In the first approach, the jobs are considered separately and for each job we have to determine which machine will process it whereas in the second approach it is the machines which are considered separately.

4.1 Job-focused approach

The job-focused approach considers jobs one by one, while it considers all m identical machines simultaneously. This approach involves three major decision steps:

- A job section rule to select the current job.
- A machine selection rule to select which machine has to assign the current selected job (the first job of the ordered list).
- An interchanging/removing rule which removes or interchanges the current job, with an already assigned job, when it is late no matter which machine is selected.

When the current job is interchanged with an already assigned job, the already assigned job takes the place of the current job in the ordered list because maybe it can be processed on an other machine. To ensure that the algorithm finishes, it should have no cycle between interchanged jobs during this step. So, when a job is interchanging with another it cannot be interchanged again while we are interchanging jobs. The schedule of assigned jobs on a machine is determined from Jeauneau and Chrétienne works [6] which propose an algorithm to solve the one machine scheduling problem with α time windows optimally in $O\left(n^3\right)$. We call this algorithm ATW (Alpha Time Windows). In their works, the schedule is decomposed into blocks $B = \{B_1, ..., B_q\}$ such as B_i is a maximal sequence of consecutive jobs (no intermediate idle time). The following notation is used to define the heuristics procedures:

- m_k The machine k.
- U The set of unscheduled jobs.
- R The set of rejected jobs.
- I The set of interchanged jobs.
- σ_k The set of jobs assigned on m_k.
- C_i The completion time of the block containing the job J_i.
- C_i^k The completion time of the block containing the job J_i for the schedule of jobs $\sigma_k \cup J_i$

Fig. 4 details the heuristic $H1E$. Then, based on the first major decision step (job selection rule), we introduce $H1L$ which uses LPT (longest processing time) rules to select the current job. For the second decision step (machine selection), Jeauneau and Chrétienne

[6] shown that there is a strong interaction between jobs of the same block, so we choice to compare the completion time of the block containing the current job for the machine selection. Moreover, select the machine k with the greatest C_c^k has the advantage to load at most as possible an interval of time of a machine and provide more space of the others machines on this interval. Hence, this strategy is most able to handle jobs having a common part of their time windows. In the three decision step, when the current job cannot be assigned to a machine without cause tardiness of a job, we choice to interchange the current job with the job which will release the most space.

Step 1. Let $R = \emptyset$ and $I = \emptyset$. Let $\sigma_k = \emptyset$, for $k = 1, ..., m$. Place the job J_i in U for $i = 1, ..., n$.
Step 2. Select a job by EDD rule (earliest due date) from U, call it J_c (the current job).
Step 3. If adding the job J_c of the set σ_k of at least one machine k gives a feasible set of jobs (all jobs can be scheduled before their due dates in the sequence obtained by ATW $(\sigma_k \cup J_c)$), then remove all jobs of I and go to Step 5.
Step 4. Let J_{x_k} the late job obtained by ATW $(\sigma_k \cup J_c)$ for the machine k. Let J_{x_y} the job such as $C_{x_y} - C_c^y = \max_{1 \leq k \leq m} \left\{ C_{x_k} - C_c^k | J_{x_k} \notin I \right\}$. If the job J_{x_y} is the same as the job J_c then put it in R, remove all jobs of I and go to Step 6. Otherwise remove the job J_{x_y} from σ_y, put it in U and put the job J_c in I.
Step 5. Remove the job J_c from U and put it in σ_b, where $C_c^b = \max_{1 \leq k \leq m} \left\{ C_c^k \right\}$ and $\sigma_b \cup J_c$ is feasible.
Step 6. If $U = \emptyset$, then append the jobs of R in any set σ_k, and stop. Otherwise, go to Step 2.

Fig. 3: Heuristic $H1E$

Each job-focused heuristic run in $O\left(mn^4\right)$ time. Indeed, there is n jobs to schedule. Each job call m times the ATW algorithm. As this algorithm run in $O\left(n^3\right)$ time, we have an overall complexity for the job-focused heuristics in $O\left(mn^4\right)$.

4.2 Machine-focused approach

The machine-focused approach is the opposite of the job-focused approach. It considers the machine one by one, while it considers the set of unscheduled jobs simultaneously. Indeed, this approach solves m times the one machine scheduling problem. The first one machine scheduling problem involves all the jobs, whereas the second machine scheduling problem is solved with the set of not yet scheduled jobs (i.e the set of jobs scheduled on the first machine are not considered), etc. This process is repeated until there is no more machine or no more unscheduled jobs. ATW algorithm is used to solve each of the m one machine scheduling problems. Among all optimum solutions, this algorithm determine the one with minimum makespan (the completion time of the scheduling is minimized). So, for each machine we know that there is no schedule which process more early jobs, but it may exist some schedules which process the same number of jobs with a higher load of the machine. Hence, after running ATW, we try to interchange some jobs in order to increase the load of the machine. Thus, after schedule a set of jobs on a machine, we try to interchange each scheduled job with each not scheduled job having a longer processing time and we test the feasibility of this interchanging with GTW algorithm (algorithm of Garey, Tarjan and Wilfong [3] which determines if a set of job is feasible and, if yes, gives a solution with minimum makespan). We obtain the heuristic $H2$ detailed in Fig. 4.

Step 1. Let $k = 1$ (The first machine). Let $R = \emptyset$ and $\sigma_k = \emptyset$, for $k = 1, ..., m$. Place the job J_i in U for $i = 1, ..., n$.
Step 2. Set $\sigma_k = ATW\,(U)$.
Step 3. If the jobs in σ_k are all early, then go to Step 5.
Step 4. Remove all late jobs from σ_k and put them in U.
Step 5. If $k < m$ and $U \neq \emptyset$ then
  Step 5.1. Order the jobs in U by LPT rule and let $i = 1$ the index of the first job in this reordered set.
  Step 5.2. Let $\sigma_k^{'}$ the set of jobs of σ_k ordered by SPT rule and let $j = 1$ the index of the first job in this set.
  Step 5.3. If $GTW\left(\sigma_k^{'} - \left\{\sigma_k^{'}[j]\right\} + \{U[i]\}\right)$ is feasible, then add $U[i]$ in σ_k and remove it from U, remove $\sigma_k^{'}[j]$ from σ_k and insert it in U with respect to LPT order, and go to Step 5.2.
  Step 5.4. If $j < |\sigma_k^{'}|$, then set $j = j + 1$ and go to Step 5.3.
  Step 5.4. If $i < |U|$, then set $i = i + 1$ and go to Step 5.2.
  Step 5.5. Set $k = k + 1$ and go to Step 2.
Step 6. Append the jobs in U to any machines and stop.

Fig. 4: Heuristic $H2$

The heuristic H2 tries to minimize the number of late jobs machine by machine. By increasing the load on machine k, the next machine $k + 1$ will be able to process more early jobs. As the interchanging of jobs is used to increase the potential of processing more early jobs for the next machine, it is not relevant to do it for the machine m (the last machine). Therefore, we not apply it on the last machine. Moreover, when we interchange jobs, we increase the load of the machine. So, it is not necessary to retry to interchange jobs which have a longest processing time than the current interchanged job (if these jobs could have been interchanged, it will be done during their interchanging steps). Therefore, in Step 5.3 of H2, we not restart the interchanging rule from Step 5.1.

The machine-focused heuristic call m times the ATW algorithm to initiate the set of scheduled jobs on the current machine. Then, we apply the interchanging rule to this machine, it call at most n^2 times the GTW algorithm. As GTW runs in at most $(n \log n)$ steps, so H2 has an overall complexity in $O\left(mn^3 \log n\right)$ time.

5 Experimental results

Table 1: Results by machine machine level

m	$H1E$	$H1L$	$H2$	$H3$	LB	Avg. Error (%)
1	**4.3**	4.59	**4.3**	4.3	4.1	4.88
2	8.04	8.26	**7.76**	7.7	7.24	6.35
5	19.08	18.86	**17.83**	17.76	16.54	7.38
10	37.92	36.63	**34.96**	34.85	32.42	7.5
15	55.39	52.84	**50.68**	50.52	46.83	7.87
20	74.66	70.71	**68.16**	67.89	63.07	7.64

We develop a simulation experiment and the results are presented in this section. The number of machines is specified at six levels: 1, 2, 5, 10, 15 and 20 whereas the number of jobs is defined by the ration n/m. This ratio is set at four levels: 5, 10, 15 and 20. Processing

times are assumed to be $U(1, 99)$. Release dates follow $U(1, n\overline{p}/(mq))$, where $\overline{p}$ is the mean processing time, and q represents the level of congestion. The values of q are set at eight levels: 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5 and 5.0. Ho and Chang [5] shown that this congestion indicator is useful in shop congestion. The shop congestion increases with q and the number of late jobs will be higher. The positive constant α of the problem is randomly computed from an uniform distribution $(1, n\overline{p}/(mq))$. As each job has to satisfy the relation $d_i = r_i + p_i + \alpha$, we calculate the due date d_i of each job from others parameters. We generate 50 instances of each problem set. Therefore, the total number of problems generated is 9,600. The heuristics $H1E$, $H1L$, $H2$ and the lower bound LB are implemented in JAVA in an Intel Core i7 2,4 GHz processor.

Table 2: Results by ratio n/m level

m	n/m	$H1E$	$H1L$	$H2$	$H3$	LB	Avg. Error (%)
1	5	**1.87**	2.07	**1.87**	1.87	1.79	4.47
	10	**3.54**	3.82	**3.54**	3.54	3.36	5.36
	15	**5.1**	5.43	**5.1**	5.1	4.87	4.72
	20	**6.68**	7.05	**6.68**	6.68	6.4	4.37
2	5	3.26	3.49	**3.1**	3.07	2.76	11.23
	10	6.54	6.75	**6.27**	6.22	5.8	7.24
	15	9.51	9.72	**9.17**	9.11	8.6	5.93
	20	12.85	13.08	**12.49**	12.4	11.8	5.08
5	5	7.6	7.86	**6.93**	6.87	5.71	20.31
	10	15.16	14.93	**13.99**	13.9	12.8	8.59
	15	23.41	23.07	**21.98**	21.9	20.59	6.36
	20	30.14	29.58	**28.44**	28.35	27.06	4.76
10	5	14.61	14.3	**12.82**	12.72	10.32	23.25
	10	30.50	29.33	**27.65**	27.56	25.21	9.32
	15	44.74	42.95	**41.23**	41.1	38.88	5.71
	20	61.84	59.93	**58.15**	58	55.28	4.90
15	5	21.39	20.19	**18.22**	18.1	14.31	26.49
	10	44.42	42.1	**39.94**	39.79	36.32	9.55
	15	67.09	64.17	**61.95**	61.76	57.99	6.5
	20	88.66	84.9	**82.61**	82.43	78.71	4.73
20	5	28.24	26.26	**23.92**	23.71	18.86	25.72
	10	59.38	55.43	**52.95**	52.7	48.11	9.54
	15	91.15	86.60	**83.93**	83.64	78.7	6.28
	20	119.89	114.53	**111.83**	111.49	106.59	4.6

The results are given according to the number of machines (Table 1), and then detailed according the ratio jobs/machines (Table 2) and the shop congestion level (Table 3). For each problem, the value of the lower bound LB is computed such as the values of heuristics. An additional heuristic $H3$ is proposed which takes at each times the best solution of H1E, H1L and H2. We also calculate the average deviation of $H3$ to LB, i.e $(H3/LB - 1) \times 100\%$.

These results show that heuristic $H2$ perform better than $H1E$ and $H1L$ and that $H1L$ perform better than $H1E$ for at most every cases.

Table 1 reveals that the average deviation of $H3$ to LB is equal to 4.88%, 6.35%, 7.38%, 7.5%, 7.87% and 7.64% for $1, 2, 5, 10, 15$ and 20 machines respectively. It is interesting to note that the average of $H3$ to LB stays stable when the number of machines increases. Indeed, for more than 2 machines the average deviation is around 7.5%. Moreover, when the number of machine is 1, heuristics $H1E$ and $H2$ solve the problem optimally ([6]). Therefore, in this case, we know that the lower bound LB has a deviation of 4.88% to

Table 3: Results by shop congestion level

m	q	$H1E$	$H1L$	$H2$	$H3$	LB	Avg. Error (%)
1	1.5	**1.36**	1.48	**1.36**	1.36	1.19	14.28
	2.0	**2.59**	2.79	**2.59**	2.59	2.39	8.37
	2.5	**3.59**	3.87	**3.59**	3.59	3.39	5.9
	3.0	**4.38**	4.7	**4.38**	4.38	4.18	4.78
	3.5	**4.92**	5.25	**4.92**	4.92	4.74	3.8
	4.0	**5.47**	5.83	**5.47**	5.47	5.28	3.6
	4.5	**5.87**	6.24	**5.87**	5.87	5.67	3.53
	5.0	**6.2**	6.58	**6.2**	6.2	6.01	3.16
2	1.5	2.54	2.49	**2.29**	2.2	1.78	23.59
	2.0	4.01	3.88	**3.65**	3.54	3.18	11.32
	2.5	6.53	6.67	**6.26**	6.19	5.74	7.84
	3.0	8.11	8.41	**7.77**	7.73	7.28	6.18
	3.5	9.44	9.76	**9.19**	9.19	8.69	5.75
	4.0	10.38	10.73	**10.12**	10.07	9.57	5.22
	4.5	11.15	11.57	**10.84**	10.81	10.34	4.54
	5.0	12.16	12.59	**11.95**	11.91	11.34	5.03
5	1.5	4.41	3.73	**3.63**	3.43	2.63	30.42
	2.0	10.93	10.02	**9.5**	9.33	7.96	17.21
	2.5	15.43	15.03	**14.25**	14.15	12.93	9.43
	3.0	19.05	18.82	**17.78**	17.74	16.55	7.19
	3.5	22.29	22.21	**20.99**	20.94	19.69	6.35
	4.0	24.69	24.83	**23.41**	23.37	22.1	5.75
	4.5	26.81	26.9	**25.35**	25.35	24.07	5.32
	5.0	29.01	29.34	**27.74**	27.73	26.4	5.04
10	1.5	8.02	**6.39**	6.50	6.02	4.54	32.6
	2.0	19.9	17.16	**16.58**	16.32	14.09	15.83
	2.5	30.89	29.03	**27.8**	27.7	25.48	8.71
	3.0	38.07	36.68	**34.72**	34.71	32.44	7
	3.5	44.95	43.67	**41.47**	41.43	39	6.23
	4.0	50.41	49.77	**47.31**	47.29	44.67	5.86
	4.5	53.56	52.67	**50.32**	50.3	47.47	5.96
	5.0	57.96	57.62	**55**	54.98	51.68	6.35
15	1.5	11.31	**8.18**	8.54	7.92	5.92	33.78
	2.0	27.50	22.81	**22.49**	22.02	19.05	15.59
	2.5	44.07	40.81	**39.245**	39.13	35.91	8.97
	3.0	55.37	52.87	**50.42**	50.37	47.29	6.51
	3.5	64.905	62.81	**59.93**	59.9	56.19	6.6
	4.0	73.81	72.03	**68.77**	68.76	64.5	6.6
	4.5	81.03	79.66	**76.16**	76.15	71.11	7.08
	5.0	85.1	83.56	**79.89**	79.89	74.68	6.98
20	1.5	15.44	**10.94**	11.50	10.69	8.32	28.49
	2.0	37.25	**30.3**	30.76	29.65	26.17	13.3
	2.5	60.65	56	**54.27**	54.12	49.54	9.24
	3.0	74.44	70.24	**67.41**	67.38	63.11	6.77
	3.5	90.14	86.84	**83.12**	83.1	78.19	6.28
	4.0	99.05	96.18	**91.85**	91.84	86.42	6.27
	4.5	107	104.47	**100.14**	100.13	93.83	6.71
	5.0	113.34	110.69	**106.2**	106.18	98.90	7.36

optimum. By assumption on the fact that the quality of the lower bound not increases when the number of machines increases, then H has a deviation $\approx 3\%$ to optimum.

Table 2 give the results according the number of machines which are detailed for th ratio n/m. Expected when the ratio $n/m = 5$, results show that the average deviation of H to LB is $< 10\%$. Moreover, if the quality of the bound not increases when the number of machines

increases, then H has a deviation to optimum of $\approx 4.5\%$, $\approx 2\%$ and $\approx 1\%$ for a ratio n/m equal to $10, 15$ and 20 respectively. For $n/m = 5$, results show a degradation of the average deviation when the number of machines increases. Indeed, the average deviation of H to LB is equal to 4.47%, 11.23%, 20.31%, 23.25%, 26.49% and 25.72% for $1, 2, 5, 10, 15$ and 20 machines. This degradation can be explained by: the degradation of the quality of LB when the number of machines increases for a small n/m ratio, the medium performance of heuristics when n/m is small. Globally, table 2 reveals the average deviation of H to LB decreases when the number of jobs increases.

Table 3 gives the results according the number of machines which are detailed for the shop congestion level. When $q \geq 2.5$, results show that the average deviation of H to LB is lesser than 10%. For these shop congestion levels, the gap between results of 1 machine case and 20 machines case is $\approx 4\%$. Therefore, if the quality quality of LB not increases when the number of machines increases, then we have an average deviation of H to optimum around 4%. For shop congestion levels 1.5 and 2.0, the average deviation of H to LB increases when the number of machines increases to reach a threshold of $\approx 34\%$ and $\approx 17\%$. This threshold is obtained for 15 and 5 machines respectively and then the degradation decreases when the number of machines increases. It should be noted that, for the one machine case, the quality of LB is yet medium which let assume that degradation of the average deviation is due to the poor quality of LB when the shop congestion level is low.

Table 4: Average CPU time

m	CPU time of $H3$ (s)	CPU time of LB (s)
1	0.001	0.002
2	0.003	0.024
5	0.015	0.768
10	0.069	8.114
15	0.165	37.99
20	0.308	99.08

The average CPU time needed to compute LB and $H3$ is presented in table 4. These results show that the average time to compute the heuristic H3 is very low although the worth case complexity is in $O\left(mn^4\right)$. For example, the mean time needed to solve an instance of 20 machines and 400 jobs is 754 milliseconds. When we consider LB, results reveals that the average time needed to compute this lower bound increases very quickly. For example, when the number of machines is 2, LB is computed has an average CPU time of 24 milliseconds whereas when the number of machines is 20, 99 seconds are needed in average.

6 Conclusions

We have studied the m parallel machines scheduling problem with α time windows in order to minimize the number of late jobs. We have proved the NP-completeness of this problem even when the number of machines is two, when all jobs have the same release date or when all jobs have the same due date. We have developed a lower bound for this problem which can also be used for the general parallel machines scheduling problem. To solve the problem, we have proposed a set of heuristics. A simulation experiment is conducted to evaluate the effectiveness of these heuristics. The results show that the average deviation to

the lower bound is equal to 4.88%, 6.35%, 7.38%, 7.5%, 7.87% and 7.64% for $1, 2, 5, 10, 15$ and 20 machines respectively. These reveals that qualities of the lower bound and heuristics are independent of the numbers of machines and globally the average deviation to the lower bound is $\approx 7.5\%$. Moreover, for the one machine, heuristics solve the problem optimally. Therefore, if the quality of the lower bound not increases when the number of machines increases, then our solution has an average deviation to optimum around 3%. Since the average CPU time needed to solve problems with 20 machines is around 300 milliseconds, our approach is able to handle large problems in a real time environment.

Acknowledgements This research was supported by French MoD DGA/MRIS and by Thales Air Systems. We also graceful to F. Barbaresco to give us the opportunity to work on this topic and to follow us during this study.

References

1. K.R Baker, Introduction to sequencing and scheduling, 108-110, Wiley, New York (1974).
2. M.R. Garey, D.S.Johnson, Computers and Intractability: A Guide to the Theory of NP Completeness, Freeman, San Francisco (1979).
3. M.R. Garey, R.E. Tarjan, G.T. Wilfong, One-processor scheduling with Symmetric Earliness and Tardiness, Mathematics of Operations Research, 13, 330-348 (1988).
4. R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annal of Discrete Mathematics, 5, 287-326 (1979).
5. J.C. Ho, Y.-L. Chang, Heuristics for minimizing mean tardiness for m parallel machines, Naval Research Logistics, 38, 367-381 (1991).
6. V. Jeauneau, C. Chrétienne, An $O\left(n^3\right)$ algorithm for the one machine scheduling problem with α Time windows to minimize the number of late jobs, preprint.
7. H.G. Kahlbacher, T.C.E Cheng, Parallel machine scheduling to minimize costs and number of tardy jobs, Discrete Applied Mathematics, 47(2), 139-164 (1993).
8. R.M. Karp, Reducibility among combinatorial problems, 85-103, Complexity of Computer Computations, R.E. Miller, J.W. Thatcher,Eds., New York: Plenum Press (1979).
9. J.M. Moore, An n-job one-machine sequencing algorithm for minimizing the number of late jobs, Management Science, 15, 139-164 (1968).
10. J. Orlin, A faster strongly polynomial minimum cost flow algorithm, Operations Research, 41, 338-350 (1993).

MISTA 2013

Algorithms and Complexity of Student Sectioning for Existing Timetables

M. Dostert · A. Politz · H. Schmitz

Abstract We show that the following fundamental question in student sectioning can be efficiently decided in polynomial time:

> Is it possible to assign m students to k sectioned courses w.r.t. a given timetable with l timeslots, such that the individual capacities of all sections are not exceeded and no student has more than one appointment per timeslot?

Our result opens the possibility to check feasibility of candidate-timetables w.r.t. this question as part of timetabling algorithms efficiently. Based on a succinct representation of solutions, we also provide an algorithm to compute optimal assignments in $O(k^2 l^2 \log(\text{sum}_A))$ where sum_A is the sum of all specified section capacities. On the other hand, we prove that adding any single of the following constraints turns the above question into an NP-complete problem:

Course-selection constraint: Student's course-selections must be respected.
Timeslot-constraint: Students have individual timeslot restrictions.
Multiple-event constraint: Sections may have multiple events in the timetable, and there must be no timeslot clashes between all section-events for each student.

Hence our investigation gives insight into the location of the borderline between efficiently solvable and computational hard problem variations.

1 Introduction

We consider an aspect of educational timetabling that appears in schools and universities likewise. Here courses often have *sections* (sometimes called *subevents*) where the same content is taught multiple times during a week, mostly due to capacity requirements. Usually no information about student-to-section assignment is known prior to timetabling because students select courses, not a particular section of a course. Hence sectioned courses put extra challenges on timetabling algorithms because a solution is a timetable *plus* a student-to-section assignment – coming along with additional

M. Dostert, A. Politz, H. Schmitz
Trier University of Applied Sciences, Germany
Corresponding author: h.schmitz@hochschule-trier.de

constraints. The latter typically comprise constraints like maximum section-sizes and conflict-free timetables for the sectioned students. Algorithms on various models with different combinations of constraints have been proposed in the literature, among them approaches where student sectioning is carried out before [3,2,14], during [4,5,12,11] or after timetabling [1,6]. In this paper we take a careful look at sectioning problems for *existing* timetables, i.e., we assume that the timeslots for all sections are already fixed by a timetable given in the input, and we have to find an assignment of students to sections which is feasible w.r.t. this data. In this way we follow the post-timetabling approach to student sectioning and we identify the additional computational effort that originates from this approach.

Our models. We give an informal description of student-sectioning models considered in this paper and begin with the Basic Student Sectioning Problem (BSS), see Section 2 for formal definitions. Here each student is enrolled in every course. Each course consists of a number of sections, each of which has a fixed capacity and a timeslot that has already been assigned. We would like to determine the maximum number of students that can be assigned to one section of each courses in a way such that

(D) each student has at most one appointment per timeslot (*disjoint-sections constraint*), and
(M) the maximum sizes of all sections are not exceeded (*maximum-capacity constraint*).

Both constraints are very often part of student-sectioning models mentioned in the literature, e.g. [1,6,11]. Observe that already (D) and (M) catch key issues because they reflect the competing interests of two main stakeholders: Students certainly want their schedules to be conflict-free, and institutions need to consider room capacities – the latter being the main reason to have sectioned courses at all.

While the basic model BSS is well applicable in some scenarios, there may well be more constraints that emerge from practice. So we also investigate extended versions of BSS by adding the following input data and constraints:

(C) Students select their courses from a larger set of courses offered, and this selection must be respected (*course-selection constraint*).
(T) Students have individual timeslot restrictions, i.e., they must not be assigned to some explicitly excluded timeslots (*timeslot constraint*).
(E) Sections may have multiple events, i.e., the same section is assigned to more than one timeslot in the input timetable. The assignment must be conflict-free w.r.t. timeslot clashes between all pairs of section-events (*multiple-event constraint*).

We concentrate on these constraints since they (or slight variations of them) appear in most models for student sectioning mentioned in the literature, often in combined form, e.g., [4,5,2]. Adding these constraints to the basic model leads in a natural way to problem versions BSS+(C), BSS+(T) and BSS+(E), respectively.

Overview. We begin in Section 2 with a careful modelling of sectioning problems in terms of matchings in bipartite graphs, and describe feasible solutions as matchings between courses and timeslots that must satisfy additional requirements. In Section 3 we show that BSS is decidable in polynomial time, while Section 4 is devoted to hardness results for the extended problem-versions. Finally, Section 5 provides an algorithm to compute also optimal *solutions* for BSS in terms of individual student-timetables efficiently. The algorithm rests on a succinct representation of solutions and is truely polynomial even if the number of assignable students is not polynomially bounded in

the input length. We also show in the final section that minimizing the number of distinct student-timetables for BSS is hard.

On the technical side, our work is inspired by [17] where class-teacher models and course scheduling are discussed with an emphasis on graph-theoretical models. For proofs omitted in this paper we refer to our preliminary technical report [8].

2 Problem Definitions

We fix some notations and formalize our problem definitions. Unless stated otherwise, all variables $i, j, k, l, m, \ldots$ are non-negative integers. Let $[k] = \{1, \ldots, k\}$ for $k \geq 0$. Sets of courses, timeslots and students are denoted as $C = \{c_1, \ldots, c_k\}, T = \{t_1, \ldots, t_l\}$ and $S = \{s_1, \ldots, s_p\}$, respectively. If the context is clear we identify courses, timeslots and students simply by their index, e.g., some course $j \in [k]$. We model sectioning problems in terms of undirected bipartite multigraphs $G = (V, E)$ with $V = C \cup T$, i.e., E is a multiset of edges, each edge joining some node in C with some node in T. For $v \in V$ denote the number of edges incident to v by $d(v)$ as degree of v, and denote by $\Delta(G)$ the maximum degree of the nodes in G. For $c \in C$ and $t \in T$ we write $m(c, t)$ for the number of edges $e \in E$ joining c with t. A matching in G is a set of edges $M \subseteq E$ such that no two edges in M share a common node. If $|M| = k$ we say that M is a k-matching. We denote by C_M (T_M, resp.) the set of nodes from C (T, resp.) that appear in M. A set $\mathcal{M}$ of matchings is called edge-disjoint if all $M \in \mathcal{M}$ are pairwise disjoint, i.e., no instance of some edge appears in more than one M. For one of the sectioning problems we also consider hypergraphs $H = (C \cup T, E)$ where each edge e is a subset of $C \cup T$ with $|e| \geq 2$. Such a hypergraph is called bipartite if $|e \cap C| = 1$ for all e. A matching in H is a subset of pairwise disjoint edges, other notations apply likewise. For background on complexity classes and the notion of NP-completeness we refer the reader to standard textbooks, e.g. [9, 13].

Recall that in the basic model each section has exactly one event, which allows to identify the section of a course by its timeslot. So together with the section capacities we represent the input timetable as a matrix $A = (a_{i,j}) \in \mathbb{N}^{l \times k}$ where l is the number of timeslots, k is the number of courses, and $a_{i,j}$ is the maximum size of the section for course c_j scheduled in timeslot t_i. Note that $a_{i,j} = 0$ if and only if there is no section of course c_j in timeslot t_i. We denote by sum_A the sum of all entries in A. Observe that we can understand such a matrix A immediately as the adjacency-matrix of a bipartite multigraph $G(A) = (C \cup T, E)$ where E is the multiset of edges $e = \{c_j, t_i\}$ with multiplicity $m(c_j, t_i) = a_{i,j}$ for $i \in [l], j \in [k]$.

A sectioning solution from a single student's point of view, who is attending all k courses, are k timeslots $t_i \in T$ (one for each of his courses), which in turn means that this student attends the section of c_j scheduled in t_i. In terms of $G(A)$ this is just a k-matching in $G(A)$ joining all nodes $c_j \in C$ with some timeslot $t_i \in T$ – while consuming one of the $a_{i,j}$ edges $e = \{c_j, t_i\}$. Finally, an entire solution is represented as a set of k-matchings $\mathcal{M}$ which is feasible if no edge is assigned to more than one student. Together, our basic model is formally defined as follows.

Basic Student Sectioning Problem (BSS)

Input: A matrix $A = (a_{i,j}) \in \mathbb{N}^{l \times k}$ with $l \geq k$ and some m.
Question: Is there a set $\mathcal{M}$ of edge-disjoint k-matchings in $G(A)$ with $|\mathcal{M}| = m$?

Observe that we may require $l \geq k$ since otherwise no solution exists due to an unavoidable violation of (D). It is also assumed in this model that there is at most one section per course and timeslot. Note that if this is not the case, we may add the respective capacities and split the assigned students arbitrarily after sectioning. Clearly, the number of students m is bounded by the minimum sum of a column in A, but also note that m is not polynomially bounded in the input length. This is due to the fact that no individual information per student is needed in this basic model.

We now define more comprehensive models by adding constraints (C), (T) and (E) individually. For the course-selection constraint (C) we assume that there is additionally some mapping $c : S \to \mathcal{P}(C)$ specifying for each student the selected courses. To fulfill these requirements there must be some assignment f of students to matchings (i.e., to their individual timetables) such that the matched courses are the selected ones. On the other hand, we do not require any more that each matching has cardinality k.

BSS+(C)

Input: A matrix $A = (a_{i,j}) \in \mathbb{N}^{l \times k}$ with $l \geq k$ and some m. *Additionally a set of students S with $|S| \geq m$ and a mapping $c : S \to \mathcal{P}(C)$ of students to selected courses.*

Question: Is there a set $\mathcal{M}$ of edge-disjoint matchings in $G(A)$ with $|\mathcal{M}|=m$, *such that there is some bijective function $f : S \to \mathcal{M}$ with*

(C) $\qquad \forall_{s \in S} \quad [f(s) = M \Rightarrow C_M = c(s)]$?

Here we ask whether at least m students from S can be sectioned such that (C) holds. Note that f is a partial function if $|S| > m$. Moreover, if all students select all k courses then we can drop the additional input and BSS+(C) is just BSS.

Next we extend BSS by the timeslot constraint (T). Again we assume that there is some additional mapping $t : S \to \mathcal{P}(T)$ specifying for each student the available timeslots. This time we need to ensure that all timeslots in each students matching are among the available timeslots for this student. Observe that we require k-matchings again and that BSS appears as a special case.

BSS+(T)

Input: A matrix $A = (a_{i,j}) \in \mathbb{N}^{l \times k}$ with $l \geq k$ and some m. *Additionally a set of students S with $|S| \geq m$ and a mapping $t : S \to \mathcal{P}(T)$ of students to available timeslots.*

Question: Is there a set $\mathcal{M}$ of edge-disjoint k-matchings in $G(A)$ with $|\mathcal{M}|=m$, *such that there is some bijective function $f : S \to \mathcal{M}$ with*

(T) $\qquad \forall_{s \in S} \quad [f(s) = M \Rightarrow T_M \subseteq t(s)]$?

Finally, we want to model with BSS+(E) that sections may have multiple events in a given timetable. So far we have identified a section of a course by the timeslot of its event in a given timetable. Correspondingly, we now identify a section of a course by the *set of timeslots* of the events of this section in a given timetable. This allows to model solutions in terms of matchings between C and T again. Note that if two sections of the same course have the same set of timeslots then we may safely unify both sections and add their capacities. So if there are at most r sections per course then we may assume that some mapping $g : C \times [r] \to \mathcal{P}(T) \times \mathbb{N}$ specifies for course c_j and its section q with $g(c_j, q) = (T_{j,q}, a_{j,q})$ the set of timeslots $T_{j,q}$ of all events of

section number q of course c_j together with the maximum capacity $a_{j,q}$ of this section. This representation of the input timetable replaces matrix A that we used before.

The sets of timeslots lead in a natural way to the notion of a bipartite multi-hypergraph $H(g) = (C \cup T, E)$, where the set E of hyperedges of $H(g)$ contains all edges

$$e = \{c_j\} \cup T_{j,q}$$

for all courses j and their sections q, each having multiplicity $m(e) = a_{j,q}$.

BSS+(E)

Input: A set of courses C, a set of timeslots T, and some m, together with a mapping $g : C \times [r] \to \mathcal{P}(T) \times \mathbb{N}$ specifying the sets of timeslots and capacities of all sections.

Question: Is there a set $\mathcal{M}$ of edge-disjoint k-matchings in $H(g)$ with $|\mathcal{M}| = m$?

Recall that in a matching $M \in \mathcal{M}$ in hypergraph $H(g)$ all edges are pairwise disjoint, i.e., there are no timeslot clashes if we understand M as a student's timetable. So if we find m edge-disjoint matchings we can assign m students to the sections of all k courses such that (E) holds.

Together, we have defined a uniform framework to formally specify sectioning problems. The notion of bipartite multigraphs common to all problem versions allows for 'mixed' versions BSS+X for X $\subseteq \{(\mathrm{C}), (\mathrm{T}), (\mathrm{E})\}$ in a natural but precise way.

3 Complexity of the Basic Student Sectioning Problem

A special case of BSS-instances (A, m) appears when all course-nodes $c \in C$ in $G(A)$ have maximum degree. Graphs with this property are sometimes called semiregular [7]. The following property of such graphs is easy to verify and we state it for further reference.

Definition 1 Let $G = (C \cup T, E)$ be a bipartite multigraph such that $|C| \leq |T|$. Then G is *semiregular* if and only if $d(c_j) = \Delta(G)$ for all $c_j \in C$.

Proposition 1 *Let $G = (C \cup T, E)$ be a semiregular bipartite multigraph such that $|C| \leq |T|$. Furthermore, let $\mathcal{M}$ be a set of matchings in G that form a partitioning of E. If $|\mathcal{M}| = \Delta(G)$, then every $M \in \mathcal{M}$ is a maximum matching with $|M| = |C|$.*

Next we see that for every yes-instance (A, m) of BSS we find a semiregular subgraph of degree m in $G(A)$, and vice versa.

Lemma 1 *For all $A \in \mathbb{N}^{l \times k}$ with $l \geq k$ and $m \geq 0$ it holds that $(A, m) \in$ BSS if and only if $G(A)$ has a semiregular subgraph $G' = (C \cup T, E')$ with $\Delta(G') = m$.*

Proof Let (A, m) be some BSS-instance and denote by $G = G(A) = (C \cup T, E)$ its graph representation with $k = |C| \leq |T| = l$. If $(A, m) \in$ BSS then there exists a set $\mathcal{M}$ containing m edge-disjoint k-matchings of $G(A)$. Now define $G' = (C \cup T, E')$ with $E' = \bigcup_{M \in \mathcal{M}} M$. Then $d(c) = m$ for every node $c \in C$ in G' because each matching M adds one edge incident to c. Furthermore, since every $M \in \mathcal{M}$ contains at most one edge incident to some $t \in T$, we also have $d(t) \leq m$ for all $t \in T$ in G'. Therefore it holds that $\Delta(G') = m$.

Conversely, assume that there is some semiregular subgraph $G' = (C \cup T, E')$ of $G(A)$ with $\Delta(G') = m$. So in G' we have $d(c) = \Delta(G')$ for any $c \in C$. We apply König's Lemma to G' and see that it can be partitioned into $\Delta(G')$ edge-disjoint matchings $\mathcal{M}$. We know from Proposition 1 that every $M \in \mathcal{M}$ is a k-matching and since $m = \Delta(G')$ we have $(A, m) \in \text{BSS}$. □

The lemma shows that the problem of deciding whether m students can be sectioned into the k courses given in timetable A is equivalent to finding a semiregular subgraph in $G(A)$ that contains all courses and has maximum degree m. So the next question is how we can determine whether or not $G(A)$ has such a subgraph. Obviously, the answer is yes if $G(A)$ itself is semiregular.

Corollary 1 *Let $A \in \mathbb{N}^{l \times k}$ with $l \geq k$ and $m \geq 0$. If $G(A)$ is semiregular then $(A, m) \in \text{BSS}$ for all $m \leq \Delta(G(A))$.*

Note that it is easy to check whether $G(A)$ is semiregular when A is given: The sums of all columns in A ($= d(c)$ for all $c \in C$) must be equal, and the sums of all rows ($= d(t)$ for all $t \in C$) must not exceed this value. If $G(A)$ is semiregular and we assign $m = \Delta(G(A))$ students to the sections of the k courses, then we have the special case that no capacities $a_{i,j}$ remain unused. We show next how to determine for arbitrary instances whether $G(A)$ has a suitable subgraph with help of a flow network.

Definition 2 Let (A, m) be some instance of BSS. Then $F(A, m) = (V, E, c)$ is a flow network with source v_s and sink v_t defined as

$$V = C \cup T \cup \{v_s, v_t\} ,$$
$$E = C \times T \cup \{v_s\} \times C \cup T \times \{v_t\} \text{ and capacities}$$
$$c(u,v) = \begin{cases} a_{i,j} & \text{if } u = c_j \wedge v = t_i \\ \min(m, \sum_{i=1}^{l} a_{i,j}) & \text{if } u = v_s \wedge v = c_j \\ \min(m, \sum_{j=1}^{k} a_{i,j}) & \text{if } u = t_i \wedge v = v_t \end{cases}$$

Observe that $F(A, m)$ can be constructed efficiently from (A, m). Now if $(A, m) \in \text{BSS}$, then we can easily obtain from the subgraph G' of $G(A)$ (see Lemma 1) a maximum flow with value $k \cdot m$ in $F(A, m)$. Conversely, if $F(A, m)$ has a flow with this value, then there is a flow of m units to each course node in C. So the saturated edges between C and T constitute a semiregular subgraph of degree m in $G(A)$, and hence $(A, m) \in \text{BSS}$ again by Lemma 1. We omit further proof details and state the following lemma.

Lemma 2 *Let (A, m) be a* BSS*-instance with $A \in \mathbb{N}^{l \times k}$ and $l \geq k$, and let f denote some maximum flow in $F(A, m)$ with value $v(f)$. Then*

$$(A, m) \in \text{BSS} \Leftrightarrow v(f) = k \cdot m \ .$$

We summarize our investigations with the decision procedure BSS$(\cdot)$ given in the following listing. Observe that $F(A, m)$ has $O(k + l)$ nodes and $O(kl)$ edges. The algorithm of Sleator and Tarjan [16] has running time $O(kl(k + l) \log(k + l)) \subseteq O(k^2 l^2 \log(k + l))$. Since the bit-length of the input is at least $k \cdot l$, the overall running time is polynomial in the input length.

Theorem 1 BSS *is in* **P**.

Input: An instance (A, m) of BSS with $A \in \mathbb{N}^{l \times k}$, $l \geq k$ and $m \geq 0$.
Output: False, if $(A, m) \notin$ BSS, and a witnessing maximum flow in $F(A, m)$ otherwise.

```
1: function BSS(A, l, k, m)
2:     F ← MAKE_FLOWNETWORK(A, m, l, k)          ▷ According to Def. 2
3:     f_max ← MAX_FLOW(F, v_s, v_t)             ▷ E.g. by the algorithm of Sleator and Tarjan [16]
4:     if v(f_max) ≠ k * m then                  ▷ Correctness follows from Lemma 2
5:         return False
6:     return f_max
```

Now consider that for any semiregular subgraph G' of $G(A)$ we have $\Delta(G') \leq \min_{j \in [k]}(\sum_{i \in [l]} a_{i,j}) \leq \text{sum}_A$ because every student has to be assigned to all courses. Thus we may compute the maximum number m^* of assignable students for a given timetable A in polynomial time by a binary search over the interval $m \in [\text{sum}_A]$ with repeated calls to the previous algorithm. To compute solutions in terms of timetables for these students (= k-matchings in $G(A)$) efficiently is more subtle because the number of matchings is usually not polynomially bounded. We come back to this point in Section 5.

4 Complexity of Adding Constraints

We show NP-hardness of the extended BSS-versions with polynomial-time many-one reductions from the NP-complete problem 3-Dimensional Matching [10]. For finite, disjoint sets X, Y and Z we say that $M \subseteq X \times Y \times Z$ is a 3-dimensional matching if for all distinct triples $(x_1, y_1, z_1), (x_2, y_2, z_2) \in M$ it holds that $x_1 \neq x_2$, $y_1 \neq y_2$ and $z_1 \neq z_2$. It is known that 3-Dimensional Matching is NP-complete even in the special case when $|X| = |Y| = |Z| = u$ and M is a perfect matching with $|M| = u$.

3-Dimensional Matching (3-DM)

Input: Finite and disjoint sets X, Y, Z with $|X| = |Y| = |Z|$ and a subset $J \subseteq X \times Y \times Z$.
Question: Is there a perfect 3-dimensional matching $M \subseteq J$?

In all proofs of the following lemmas let $X = \{x_i \mid i \in [u]\}$, $Y = \{y_i \mid i \in [u]\}$, $Z = \{z_i \mid i \in [u]\}$ for some $u \geq 1$ and $J \subseteq X \times Y \times Z$ be an arbitrary instance of 3-DM. We only state the definition of the reduction function in each case.

Lemma 3 3-DM *is polynomial-time reducible to* BSS+(C).

Proof The idea of the reduction is to encode each of the sets X, Y, Z separately as a set of u courses, each of them having a single section with capacity 1. All the sections of the courses for X go to timeslot t_1, all sections of Y to timeslot t_2 and all sections of Z to t_3. This can be achieved if we define $A = (a_{i,j})^{3 \times 3u}$ with $a_{i,j} = 1$ for $i \in [3]$ and $(i-1) \cdot u + 1 \leq j \leq i \cdot u$, and $a_{i,j} = 0$ otherwise. So $G(A)$ is a bipartite graph with node partitions $C = \{c_j \mid j \in [3u]\}$ and $T = \{t_i \mid i \in [3]\}$. Each $(x, y, z) \in J$ is regarded as a student with course-selection (x, y, z). So we set $S = J$ and define $c(s) = (c_p, c_{u+q}, c_{2u+r})$ for all $s = (x_p, y_q, z_r) \in S$. Note that the courses for X, Y and Z are disjoint due to offsets u and $2u$. Moreover, we ask whether $m = u$ students can be sectioned. Clearly, the reduction function h with $h(X, Y, Z, J) = (A, u, J, c)$ is computable in polynomial time. □

Observe that the reduction function maps only to instances where each course has just one section and where no timeslot clashes can occur, i.e., the disjoint-section constraint (D) is always fulfilled. So to obtain NP-hardness it is enough if (M) and (C) are present, and if students are limited to choose three courses - even if there are no sectioned courses at all.

Lemma 4 3-DM *is polynomial-time reducible to* BSS+(T).

Proof The idea of the reduction is very similar to the previous proof, we just exchange the roles of courses and timeslots. So A is the transposed matrix of the one before and $G(A)$ has node partitions $C = \{c_1, c_2, c_3\}$ and $T = \{t_i | i \in [3u]\}$. Each $(x, y, z) \in J$ is then regarded as a student again, whose available timeslots are the timeslots of sections x, y, and z. So we set $S = J$ and define $t(s) = (t_p, t_{u+q}, t_{2u+r})$ for all $s = (x_p, y_q, z_r) \in S$. Again, the reduction function h with $h(X, Y, Z, J) = (A, u, J, t)$ is computable in polynomial-time. □

Notice that this time h maps only to instances with 3 courses and again no timeslot clashes can occur. So to obtain NP-hardness it is enough if (M) and (T) are present, and if the number of courses is limited to three.

Lemma 5 3-DM *is polynomial-time reducible to* BSS+(E).

Proof The reduction encodes X as a set of u courses and the sets Y and Z are regarded as two disjoint sets of u timeslots. Each triple $(x, y, z) \in J$ represents a section of course x with two events scheduled in timeslots y and z and having capacity 1. Hence the number of sections of some course x is equal to the number of triples in J with x as first component. For all $x_p \in X$ and $i \in [v]$ with $v = |\{(x, y, z) \in J \mid x = x_p\}|$ we set $g(c_p, i) = (\{t_q, t_{u+m}\}, 1)$ if (x_p, y_q, z_m) is the i-th triple in J with first component x_p. Observe that in this case $H(g)$ is a bipartite hypergraph with node partitions $C = \{c_j \mid j \in [u]\}$ and $T = \{t_i \mid i \in [2h]\}$. We define the reduction function h as $h(X, Y, Z, J) = (X, Y \cup Z, 1, g)$ which asks whether a single student can be sectioned, and which is computable in polynomial time. □

This time the reduction function h maps only to instances where each section has exactly two events and only one student is considered, i.e., the maximum-capacity constraint (M) can not be violated for the existing sections. So to obtain NP-hardness it is enough if (D) and (E) are present, if the number of events per section is limited to two and if a single student needs to be sectioned.

Since we can check a solution in polynomial time, we see that **NP** is an upper bound for the complexity of our decision problems. This still holds if we combine our constraints, so we have:

Theorem 2 *All problems* BSS+X *for non-empty* X$\subseteq \{\mathrm{C}, \mathrm{T}, \mathrm{E}\}$ *are NP-complete.*

5 Computing Succinct Solutions for BSS

We investigate in this section how to efficiently compute solutions for BSS in terms of individual timetables for the optimal number of students. We have already mentioned in Section 2 that we can interpret a k-matching $M \in \mathcal{M}$ in $G(A)$ as such a single student's timetable. Let us first recall from Section 3 what we can do efficiently so far if the input timetable A is given:

– Compute the optimal number of students m^* with a binary search over m and subsequent calls to algorithm BSS($\cdot$) stated after Theorem 1.
– Get the maximum flow $f^*_{\max}$ in $F(A, m^*)$ from this algorithm.

Furthermore, we can construct from $f^*_{\max}$ a submatrix A' of A such that $G(A')$ is semiregular and $m^* = \Delta(G(A'))$. For this we just set

$$a'_{i,j} = f^*_{\max}(c_j, t_i)$$

for all $(c_j, t_i) \in C \times T$. The advantage of A' over the original A is that the solution $\mathcal{M}$ we are looking for is a partitioning of the edges in $G(A')$, i.e., no capacities in A' remain unused. It is easy to see that in this case edge-partitioning is the EDGE-COLORING problem with $\Delta(G(A'))$-many colors (cf. Proposition 1), so we could fall back on some standard algorithm for the latter, see, e.g. [15].

However, there is a major drawback with this approach if complexity is considered. This is because all known EDGE-COLORING algorithms have factor $|E|$ in their running time, but the number of edges in $G(A')$ is not polynomially bounded in the bit-length of input A. Note that there can be as many as $O(\text{sum}_A)$ edges which can only be bounded pseudo-polynomially in the input length. This is a substantial aspect since in practice rather large sums of capacities occur. So our goal in this section is to find an algorithm for optimal solutions that is truly polynomial in the bit-length of input A. For this, a succinct representation of solutions $\mathcal{M}$ is needed.

It will turn out that there is always an optimal solution that has only a small number of *different* student timetables $M \in \mathcal{M}$. Together with the fact that in the BSS setting students are indistinguishable, it will be enough to compute the number of students that follow each of these timetables. We say that two matchings $M, M' \in \mathcal{M}$ in multigraph $G(A)$ are *distinct* if they do not comprise the same set of edges.

Definition 3 (Succinct solution) Let A be some BSS-matrix such that $G(A)$ is semiregular and let $\mathcal{M}$ be a set of distinct k-matchings in $G(A)$. A function $f : \mathcal{M} \to \mathbb{N}$ is called a *succinct solution* for A, if $\sum_{M:\{c_j,t_i\}\in M} f(M) = a_{i,j}$ for all $j \in [k]$, $i \in [l]$.

This condition ensures that the sum of all students that follow the distinct student timetables in $\mathcal{M}$ is exactly the maximum number of students that can be assigned to A. We now state an algorithm OPT_SOLUTION to compute such a succinct solution efficiently in the following listing. Note that the algorithm is underspecified in the sense that it is not clear yet how to compute in line 9 a matching that saturates all nodes of maximal degree. We come back to this in the theorem below. To see the following lemma one can show by induction on the maximal degree $\Delta(G(A))$ in line 8 that the while-loop computes a succinct solution for A if G(A) is semiregular.

Lemma 6 *Algorithm* OPT_SOLUTION *is correct and the while-loop in line* 8 *has at most* $k(l+1)$ *iterations.*

Theorem 3 *Let A be a* BSS*-timetable with $A \in \mathbb{N}^{l\times k}$ and $l \geq k$. Then Algorithm* OPT_SOLUTION *computes a succinct solution for the optimal number of assignable students in $O(k^2 l^2 \log(\text{sum}_A))$.*

Proof The value of d in line 2 is computable in $O(kl)$ and is bounded by $O(\text{sum}_A)$. So by Theorem 1 the binary search in line 3 can be carried out in $O(\log(\text{sum}_A)k^2l^2\log(k+l))$. Lines 4 to 7 are in $O(kl)$ and we know from the previous lemma that the while-loop has

Input: A BSS-timetable A with $A \in \mathbb{N}^{l \times k}$ and $l \geq k$.
Output: A succinct solution f for the maximum number of assignable students.

```
1: function OPT_SOLUTION(A, l, k)
2:     d ← min_{j∈[k]}(Σ_{i∈[l]} a_{i,j})                    ▷ upper bound for number of students
3:     m*, f*_max ← BIN_SEARCH(0 ≤ m ≤ d, BSS(A, l, k, m))   ▷ optimal number of students
4:     A ← 0_{l×k}
5:     for each (c_j, t_i) in C × T                          ▷ reduce matrix to semiregular subgraph
6:         a_{i,j} ← f*_max(c_j, t_i)
7:     f ← ∅
8:     while Δ(G(A)) ≠ 0 do
9:         M ← some k-matching in G(A) saturating all nodes of maximal degree
10:        n ← min(min_{{c_j,t_i}∈M}(a_{i,j}), min_{t_i∈T\T_M}(Δ(G(A)) − d(t_i)))
11:        for each {c_j, t_i} in M
12:            a_{i,j} ← a_{i,j} − n
13:        f(M) ← n
14:    return f
```

at most $k(l+1)$ iterations. Next we need to argue how we can find matching M in line 9. Since we only need a single k-matching we can construct a graph G' from A which has a single edge (i,j) if $a_{i,j} > 0$. Note that the number of edges in G' is bounded by $O(kl)$ as opposed to the number of edges in $G(A')$. We use the algorithm by Cole and Hopcroft [7] to find a k-matching in G' that saturates all nodes of maximum degree. The running time of this algorithm on input G' is bounded by $O(kl \log k)$. The other statements in the body of the while-loop have smaller running times, so the while-loop is in $O(k^2 l^2 \log k)$. Hence the overall running time is dominated by the binary search which can be simplified to $O(k^2 l^2 \log(\mathrm{sum}_A))$. □

Observe that the given bound is polynomial in the bit-length of A. But the theorem also gives some insight into the structure of BSS-solutions: Although the number of sectioned students can be large, we can efficiently compute an optimal solution with not more than $k(l+1)$ many distinct student-timetables.

Corollary 2 *For any* BSS*-timetable A there exists an optimal succinct solution $f : \mathcal{M} \to \mathbb{N}$ with $|\mathcal{M}| \leq k(l+1)$.*

A small number of different student-timetables is a reasonable goal in student sectioning since it has some advantages for students (study groups, car pools) and it simplifies administration. We make this question precise as follows.

MIN-BSS

Input: A matrix $A = (a_{i,j}) \in \mathbb{N}^{l \times k}$ with $l \geq k$ such that $G(A)$ is semiregular, and some $m \leq k(l+1)$.

Question: Is there a succinct solution $f : \mathcal{M} \to \mathbb{N}$ for A with $|\mathcal{M}| = m$?

It seems that the problem of finding a solution with a *minimum* number of different timetables is much harder than finding a solution with a *small* number. Again, we shorten the proof and only state the definition of the reduction function.

Theorem 4 MIN-BSS *is NP-complete.*

Proof Observe that we can nondeterministically guess all sets $\mathcal{M}$ of k-matchings of size m and assign some value $f(M) \leq \mathrm{sum}_A$ to all $M \in \mathcal{M}$. To check whether the

condition from the definition of succinct solutions holds for this choice of f can be done deterministically in polynomial time, so MIN-BSS is in **NP**. We show hardness with a polynomial-time many-one reduction from the NP-complete problem SUBSET SUM [9]. Let (a, k) be some instance of this problem with $a = (a_1, \ldots, a_m) \in \mathbb{N}^m$, $a_i > 0$ and $k \in \mathbb{N}$. Then (a, k) is a yes-instance if and only if there exists some $I \subseteq [m]$ with $\sum_I a_i = k$. We may assume w.l.o.g. that $0 < k \leq s = \sum_{i \in [m]} a_i$ since otherwise (a, k) is trivial.

We define a reduction function h as $h(a, k) = (B, m)$ such that $B \in \mathbb{N}^{(m+2)\times 3}$ with rows b_i given by

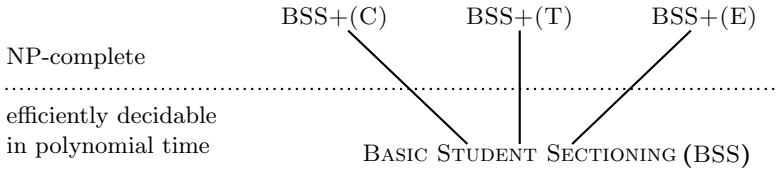
$$b_i = \begin{cases} (0, \quad a_i, \quad 0) & \text{for } i \in [m], \\ (k, \quad 0, \; s-k) & \text{for } i = m+1, \text{ and} \\ (s-k, \; 0, \quad k) & \text{for } i = m+2. \end{cases} \tag{1}$$

Then h can be computed in polynomial time and $G(B)$ is semiregular. Moreover, it holds that (a, k) is a yes-instance for SUBSET SUM if and only if (B, m) is a yes-instance for MIN-BSS. □

We see from this reduction that the problem remains NP-complete even if the number of courses is fixed to $k = 3$ in all instances.

Conclusion

With our complexity analysis of decision problems BSS+X for X$\subseteq \{(\mathrm{C}), (\mathrm{T}), (\mathrm{E})\}$ we have revealed the borderline between efficiently solvable and NP-hard combinations of constraints in student sectioning, see Theorems 1 and 2, which can be summarized as follows:



This shows that there is a basic version of student sectioning that can be used to evaluate candidate-timetables efficiently w.r.t. this model. Even if there are more side-constraints in practice, BSS can be used as a necessary condition to partially rule out infeasible solutions as part of timetabling algorithms. On the other hand we have identified the limitation of this application by our hardness results: There does not exist an efficient and sufficient algorithm to test the feasibility of assignments of students to sections during timetabling if additional constraints are present (unless **P** equals **NP**).

To the best of our knowledge, this is the first complexity analysis of student sectioning if timetables are given – with one exception: In [6] the authors sketch a proof for NP-hardness of their model from which one can conclude already that also BSS+(E) is NP-hard. In their reduction, the number of events per section is not limited. We strengthened this result for BSS+(E) in Lemma 5 and proved that it is NP-complete even if sections only have two events.

Finally we have shown that even if the input for BSS is not proportional to the number of assignable students (as is the case with input timetable A), it is still possible

to compute optimal solutions in polynomial time. This led to an efficient algorithm which outputs a *small* number of different student timetables. However, to determine the minimum number of different student timetables for BSS is again a hard problem.

There are at least two directions for further research one can follow. On one hand, we see that our reductions leave space for interesting special cases that may be also solvable in polynomial time. Here we think of restricted forms of additional constraints that may well appear in practice, e.g., if only a small constant number of courses have sections with multiple events. On the other hand, our results justify to attack sectioning problems with help of inexact algorithms, and it would be interesting to find efficient approximation algorithms with proven error-bounds for the respective graph-based models of student sectioning.

Acknowledgements We thank our anonymous referees for their valuable comments.

References

1. Alvarez-Valdes, R., Crespo, E., Tamarit, J.M.: Assigning students to course sections using tabu search. Annals of Operations Research **96**(1), 1–16 (2000)
2. Amintoosi, M., Haddadnia, J.: Feature selection in a fuzzy student sectioning algorithm. Practice and Theory of Automated Timetabling V pp. 147–160 (2005)
3. Amintoosi, M., Yazdi, H.S., Haddadnia, J.: Fuzzy student sectioning. PATAT'04: Proceedings of the 5th international conference on Practice and Theory of Automated Timetabling pp. 421–425 (2004)
4. Aubin, J., Ferland, J.A.: A large scale timetabling problem. Computers and Operations Research **16**(1), 67–77 (1989)
5. Carter, M.W.: A Comprehensive Course Timetabling and Student Scheduling System at the University of Waterloo. In: PATAT '00: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III. Springer-Verlag (2000)
6. Cheng, E., Kruk, S., Lipman, M.: Flow formulations for the student scheduling problem. Practice and Theory of Automated Timetabling IV pp. 299–309 (2003)
7. Cole, R., Hopcroft, J.: On edge coloring bipartite graphs. SIAM Journal on Computing (1982)
8. Dostert, M., Politz, A., Schmitz, H.: Student Sectioning for Fixed Timetables. Tech. Rep. 2013-1, Hochschule Trier, Germany (2013). URL http://www.hochschule-trier.de/index.php?id=informatikberichte
9. Garey, M.R., Johnson, D.S.: Computers and Intractability (1980)
10. Karp, R.M.: Reducibility among combinatorial problems. Complexity of Computer Computations (1972)
11. Müller, T., Murray, K.: Comprehensive approach to student sectioning. Annals of Operations Research **181**(1), 249–269 (2010)
12. Müller, T., Rudová, H., Barták, R.: Minimal perturbation problem in course timetabling. In: PATAT'04: Proceedings of the 5th international conference on Practice and Theory of Automated Timetabling. Springer-Verlag (2004)
13. Papadimitriou, C.H.: Computational complexity. Addison Wesley (1994)
14. Schmitz, H., Heimfarth, C.: Cross-Curriculum Scheduling with Themis. PATAT'10: Proceedings of the 8th international conference on Practice and Theory of Automated Timetabling pp. 385–391 (2010)
15. Schrijver, A.: Bipartite Edge Coloring in $O(\Delta m)$ Time. SIAM Journal on Computing (1998)
16. Sleator, D.D., Tarjan, R.E.: A data structure for dynamic trees. Journal of computer and system sciences (1983)
17. de Werra, D.: An introduction to timetabling. European Journal of Operational Research **19**(2), 151–162 (1985)

MISTA 2013

Balancing and Sequencing Mixed-Model Assembly Lines to Minimize Work Overload in Real World Workstations

Luiza Zeltzer, Veronique Limère and El-Houssaine Aghezzaf

Abstract In mixed-model line balancing, we usually have overload after having optimally balanced the line. To further reduce this overload, sequencing of the models may be required. This paper presents two Mixed Integer Linear Program (MILP) models, where one balances the overload and the second sequences models to minimize the overload. Two heuristics are developed to solve some realistic cases. The solutions representation, algorithms and results are systematically described. The efficiency of both models and heuristics are examined by applying them to real world data.

1 Introduction

Over the last decades, product variety in automotive industry has increased drastically. This high variety is driven by both market demand and sustainability requirements; consequently it makes mixed-model assembly lines very common. A mixed-model assembly line assembles different products in the same line and is typically characterized by two correlated problems: balancing and sequencing.

Assembly line balancing problem (ALBP) considers task operation times to balance the work load and it involves defining the assignment of tasks to different workstations. Considering real world scenarios, optimizing the balance can be extremely demanding and complex. It occurs as a result of a high number of tasks and workstations in the line. Taking it into consideration, generating a solution for an ALBP might be a long-term decision problem. Assembly line sequencing problem (ALSP) considers the difference between models to sequence the products to be assembled [1]. Models sequencing applies a detailed schedule and it is extremely important to minimize the overload that still might occur after the balance.

The solutions for ALBP and ALSP are complementary. This paper focuses on minimizing the work overload at each workstation of a mixed-model assembly line. Initially, an optimization Mixed Integer Linear Program (MILP) model is developed to determine a balance that minimizes the overload at each workstation. Since this is high time-consuming solution, an algorithm of heuristic nature is also proposed. Finally, sequencing is used to minimize this overload.

2 Brief Literature Review

The assembly line balancing problem (ALBP) consists in optimizing the partitioning of assembly work among workstations[2] based on a specific objective. Research on the ALBP has traditionally focused on the simple assembly line balancing problem (SALBP). Recently, however, significant research effort [3] has been made in modelling and solving more realistic generalized problems (GALBP). A line, where only one kind of product is assembled is called single-model line; when several models can be assembled on the same line, the assembly line balancing problem is called mixed-model line.

Mixed-model assembly lines are becoming very frequent [4] due to the increase in product variety, e.g. automotive industry. This increase is necessary to answer the market and sustainability demands. Meanwhile different models may have different assembly requirement and consequently different operation times. The sequence

that models are launched on the line has a direct impact on workstations overload. A work overload situation [5] occurs whenever several time-demanding models follow each other successively on the line, which means the operator is not able to finish the work into the cycle time.

The assembly line sequencing problem (ALSP) consists [6] in sequencing different models in the assembly line in order to avoid work overload at the workstations. There are mainly two approaches to deal with ALSP, the first focuses on stopping the line and compensating the overload situation and the second focuses on procedures to avoid this situation. The main objective of the first research line is to minimize line stoppage times, it takes into account different factors as human factor policies[7], computational complexity of the sequencing problem[8] and use of algorithms as branch-and-bound and simulating annealing [9]. The second research line [10] [11] uses typical variables of the line to optimize the workstation usage: as assembly line station length, assembly time and cycle time.

The paper proposes solutions for ALBP and ALSP aiming to minimize work overload situations. Section 3 specifies the mixed-model assembly line balancing model problem. Section 4 describes the proposed solution to minimize the work overload. Subsection 4.1 reviews the MILP for ALBP, subsection 4.2 presents the heuristics developed for ALBP, subsection 4.3 reviews the sequencing model used and subsection 4.4 describes the heuristics proposed for ALSP. Section 5 specifies and analyses the results. Section 6 concludes the work done and presents future work.

3 Problem Statement

The research carried out was developed in collaboration with the automotive industry and their suppliers. The production units under study are responsible for the production of vehicles and original equipment's (OEM). As referred previously, the present work focuses on minimizing the work overload of mixed-model assembly lines. It takes into account balancing and sequencing.

The problem is described using the following parameters.

Notations:

K	is the set of workstations, index k
J	is the set of tasks, index j
O	is the set of operators, index o
M	is the set of models, index m
$KQual_j$	is the set of qualified workstations *k* for a task *j*, index $kqual_j$
$OQual_j$	is the set of qualified operators *o* for a task *j*, index $oqual_j$
$maxOp(k)$	is the maximum number of operators in the workstation *k*
c	cycle time
t_{jm}	is the processing time of task *j* for model *m*
$Pred(j)$	is the set of direct predecessors, index of task $pred(j)$
b_m	is the demand of model *m* in the model-mix, with $\sum_{m \in M} b_m = 1$
l_k	is the length of the workstation *k* in units of time
I	is the set of units to be sequenced, index: $i=1,\ldots,I$
ρ_{ki}	is the sum of all operation tasks time of unit *i* at workstation *k*
RT_{ki}	is the remaining time in workstation *k* after assembling unit *i*

Mixed-Model Assembly Line Balancing Problem

The definition consists of (1) given a set *M* of different models, a set *K* of workstations, a set *O* of operators, a set *J* of tasks; (2) each task *j* needs to be assigned to a workstation *k* respecting the restrictions of (3) precedence relationships between tasks, (4) qualified workstations and (5) qualified operators to execute the respective task.

Each model *m* requires a subset *J* of tasks to be executed. These tasks are also called assembly operations. Each task *j* has a processing time t_{jm} according to model *m*. The processing time may be also called task time or operation time. Different models may demand different subsets of tasks. All tasks must be assigned respecting the preceding relationships, i.e. all predecessors *Pj* of task *j* must be assigned before task *j*.

Each workstation *k* is qualified to perform a subset *J* of tasks and each task *j* must be assigned to at least one workstation *k*. Each workstation *k* has also a maximum number of assigned operators *o*. Each operator *o* assigned to workstation *k* must be qualified to execute the subset *J* of tasks assigned to workstation *k*. Each workstation *k also* has a maximum number of operators *maxOp(k)*.

Mixed-Model Assembly Line Sequencing Problem

The definition consists of (1) given a set I of units to be produced, (2) each unit *i* must be launched down an assembly line in a specific sequence respecting the restrictions of (3) the length l_k of each workstation *k*, (4) the cycle time *c* and (5) the operation time ρ_{ki}.

Workstations have a different length l_k. The length l_k of each workstation is the maximum operation time of a model *m* on workstation *k*. Work overload occurs when the right border of the workstation is reached without finishing all the operations tasks assigned to workstations *k*.

4 Line Balancing and Sequencing Models and Heuristics

To solve the mixed-model ALBP and ALSP, two methods were considered: an exact method and a heuristic method. The next subsections present in detail the MILP models used and the heuristics procedures proposed.

4.1. Mixed-Model Assembly Line Balancing Problem – MILP Model

As mentioned previously, the main goal of the investigated problem is to minimize the balance loss, which is related to minimizing the total overload of each workstation and optimize the line balance. The proposed model is represented as:

Variables

$Overl_{km} \geq 0$	represents the overload on workstation *k* and model *m;*
$X_{kj} \in \{0,1\}$	is 1 if task *j* assigned to workstation *k*, 0 otherwise
$Y_k \in \{0,1\}$	is 1 if workstation *k* is open, 0 otherwise
$Op_k \in Z_+$	is the number of operators assigned to workstation *k* if open, 0 otherwise

Mathematical Model Formulation

$$Min \sum_{k \in K} \sum_{m \in M} Overl_{km} * b_m \quad \textbf{(1)}$$

subject to:

$$\sum_{k \in KQual_j} X_{kj} = 1, \qquad \forall j \quad \textbf{(2)}$$

$$\sum_{k \in K} kX_{kj} \leq \sum_{k \in K} kX_{ki}, \qquad \forall i \; and \; \forall j \in Pred(i) \quad \textbf{(3)}$$

$$X_{kj} - Y_k \leq 0, \qquad \forall k, j \quad \textbf{(4)}$$

$$Y_k - Y_{k-1} \leq 0, \qquad \forall k \geq 1 \quad \textbf{(4')}$$

$$\sum_{j \in J} t_{jm} * X_{kj} \leq Overl_{km} + c * Op(k), \qquad \forall m, k \quad \textbf{(5)}$$

$$Op_k - maxOp(k) Y_k \leq 0, \qquad \forall k \quad \textbf{(5')}$$

Equation (1) represents the objective function; it minimizes the overload of each workstation *k* for each model *m*. It takes into account the demand b_m for each model *m*.

The set of constraints are described as: Eq. (2) ensures that each task *j* is assigned to one workstation *k*, Eq. (3) determines that a task *j* is assigned only if all tasks in *Pred(j)* are assigned. Eq. (4) and (4') impose the increasing order in which workstations *k* are defined as open. Finally, the most important restriction is defined by Eq. (5): it guarantees that the load of workstation *k* for model *m is* as low as possible, i.e. the sum of all tasks assigned to workstation *k* is below the cycle time *c*; it also considers the number of operators Op_k in the workstation *k*. Finally Eq. (5') guarantees that the number of operators Op_k cannot exceed the maximum number of operators $maxOp(k)$ that can be assigned to workstation *k*.

It was possible to observe the combinatorial nature of the problem and the difficulty of obtaining an optimal solution in an adequate computational time. It is known as a NP-hard (non-deterministic polynomial-time hard) problem and it is highly time and computational consuming. Based on that an alternative heuristics method was implemented. It is important to refer that the model is extremely useful since it provides the optimal solution for the problem and it can be used to evaluate the efficiency of the heuristics procedures.

4.2. Mixed-Model Line Balancing using heuristics procedures

The MlLP model is able to provide the optimal solution for a problem but it is time and machine consuming. A set of heuristics procedures and rules have been implemented to obtain a quicker and, simultaneously, an efficient solution for real world problems. An algorithm is proposed, typically it generates an initial solution and by heuristics rules the initial solution is optimized. Those rules are systematically applied until a satisfactory solution is obtained. The next subsections detail the approach implemented.

4.2.1. Super Model

As the problem under study is a mixed-model problem, a super model *sm* is generated. The main goal is to define a model that requires task operation times as an effective representation of all models. The *sm* task time is determined as:

$$t_{jsm} = \sum_{m=1}^{M} t_{jm} * b_m + \alpha * \left(t_j^{max} - \sum_{m=1}^{M} t_{jm} * b_m \right) \qquad \textbf{(6)}$$

Where:

t_{jsm}	is the *sm* operating time of task *j*,
α	is a variant value between 0 and 1,
t_j^{max}	is the maximum task time of a task *j*.

α assumes 5 different values from 0 to 1 with 0.25 of interval. This variation produces a set of 5 different solutions. The reason to apply such variation is to compensate the different values of t_{jm}, i.e., task *t* may have very different values according to different models *m*. A satisfactory solution for all models can be obtained by compensating this difference.

4.2.2. Mixed-Model Assembly Line Balancing Algorithm

Below, we present the algorithm to solve the ALBP. It integrates the following steps:

Step 1. The *sm* task times are populated based on all t_{jm}, i.e., based on all task times for all models. The initial solution is populated with α=0.

Step 2. The crucial element to generate an initial solution is to define the order that the tasks are assigned. The tasks are assigned to the workstations in a specific order respecting 2 main restrictions: 1) the precedence relationships between tasks and 2) the ranked positional weight.

The weight of each task is defined using the method proposed by Helgenson [12]. It consists in:

1) Defining the precedence relationship diagram,
2) For each task, determining the positional weight:
   - The longest path from the first task to the last task of the network.
3) Ranking the tasks in descending order of ranked positional weight (RPW).

Step 3. Selecting workstations. Task *t* can be assigned only to qualified workstations. To define to which qualified workstation *k* task *t* is assigned a set of conditions are taken into consideration. Those conditions are mainly based on the load of the workstation and the operator assigned to the workstation. Tasks are assigned to workstations with highest under load or lowest overload.

Figure 1 represents in detail Step 1, Step 2 and Step 3.

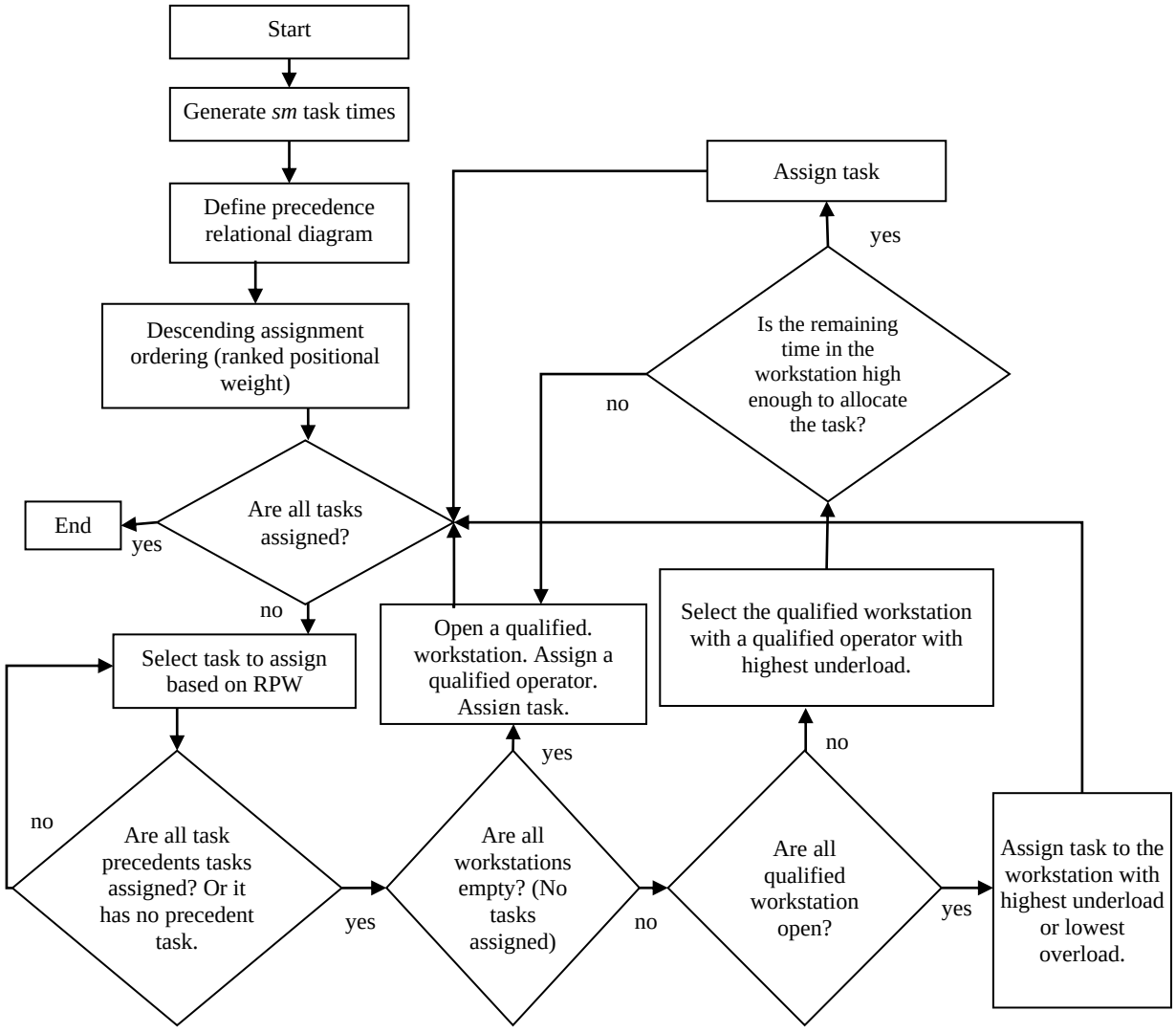


Figure 1 - Mixed-Model Assembly Line Balancing Algorithm

Step 4. The procedure described on step 3 is repeated for α=0.25, α=0.50, α=0.75 and α=1.00. The line balancing solutions obtained are stored in a set.

Step 5. Solution analysis. The best solution is selected from the 5 interactions. 2 different situations may occur: 1) there is no overload registered or 2) there is some overload registered. When no overload is produced means that we have the optimal solution. When some overload is still generated after the balance, an optimization step is done and the line is rebalanced.

Step 6. Solution optimization. The balance solution might be not satisfactory and in this case a set of optimization decisions are applied to the balance proposed. This process considers the total work overload per workstation:

$$workstation\ k\ overload = \sum_{m=1}^{M} overload\ model\ m\ \ on\ workstation\ k \qquad \textbf{(7)}$$

The total overload of the workstation *k* is the sum of the overload for each model *m* on workstation *k*. The objective is to minimize the total work overload in each workstation. Subsequently, tasks are permutated from one workstation to another following the decending assignment order. In each interaction, the new overload is calculated and the best balance stored.

4.3. Mixed-Model Assembly Line Sequencing Problem – MILP Model

The sequencing problem is typical for a mixed-model assembly line. The production sequence has a considerable impact on the total load of a workstation. Models with a high complexity factor should be followed by models with a low complexity factor, so the station load is compensated and work peaks are avoided.

The mathematical model used is based on the solution proposed by Scholl, et al. [11]. The difference is how workstation length l_k is calculated. Since we correlate mixed-model ALBP and ALSP, the length l_k of workstation *k* is the maximum operation load at the workstation, i.e., we use the result obtained by the MILP model to solve ALBP to define l_k, therefore it is equal to the maximum load that workstation *k* assumes.

The model used is represented as:

Variables

$x_{mi} \in \{0, 1\}$	is 1 if model *m* is assigned to position *i* of the sequence, 0 otherwise.
s_{ki}	represents the position in the workstation *k* when *i* starts to be processed.

Mathematical Model Formulation

$$\boldsymbol{Min} \sum_{\boldsymbol{k \in K}} \sum_{\boldsymbol{i \in I}} Overl_{ki} \qquad \textbf{(8)}$$

subject to:

$$\sum_{m \in M} x_{mi} = 1 \qquad \textbf{(9)}$$

$$\sum_{i \in I} x_{mi} = b_m \qquad \textbf{(10)}$$

$$s_{ki+1} \geq s_{ki} + \rho_{ki} - Over_{ki} - c \qquad \textbf{(11)}$$

$$s_{ki} + \rho_{ki} - Over_{ki} \leq l_k \qquad \textbf{(12)}$$

$$s_{ki} \geq 0, Over_{ki} \geq 0 \qquad \textbf{(13)}$$

$$s_{ki} = 0, s_{ki+1} = 0 \qquad \textbf{(14)}$$

Eq. (8) represents the objective function, it minimizes the total work overload of each workstation *k*. The set of constraints are described as: Eq. (9) ensures that each unit *i* is assigned to a position in the sequence. Eq. (10) ensures that all units *i* are assigned. Eq. (11) guarantees that a unit *i+1* cannot start before the finalization of unit *i*. Eq. (12) and (13) limit the load of the station to its length. Eq. (14) ensure the initial state of the workstation.

4.4. Mixed-Model Line Sequencing using heuristics procedures

The MILP model provides a solution very close to the optimal but, as the MILP to solve ALBP, it is time and machine consuming. It happens due to the fact that it is a NP-hard problem as well. We have defined a set of heuristics procedures to approach this kind of problem. Henceforth, they will be implemented.

The sequencing procedures are sets of iterations which define a specific assembling order. The main objective is to define an order in which workstation work load are optimal, i.e., workstation load is as close as possible to the cycle time. The following assumptions are considered:

- $l_k = c$ The length of each workstation k is equal to the cycle time c,
- $RT_{ki} = c - \rho_{ki}$ The remaining time RT_{ki} in workstation k, after processing unit i is equal to the cycle time i minus the total operation time ρ_{ki} of unit i,
- There is no buffer between workstations,
- Operators are fixed and assigned to specific workstations, i.e., they cannot start assembling a unit before it arrives in the workstation.

The algorithm to solve mixed-model ALSP is composed by the following steps:
Step 1. After balancing the line, order the units to be assembled according to their work overload.

Step 2. Define a descendent order the units based in their demand. If all models have the same demand, this step is not necessary.

Step 3. A unit with high overload is followed by a unit with low overload. This action is taken in order to compensate the time spent in units with different operation times.

Step 4. Consider the demand. Models with low operation time and high demand are grouped, i.e, they are assembled in a consecutive order. These units have identical assembly operations, this step allows to optimize the work time in the station.

Step 5. Optimize the number of qualified operators assigned to workstations. More operators may be assigned to workstations with high operation load and less to workstations with low load.

However the procedures were not developed, they were validated by simulating real world scenarios with real data. The previous steps enable to minimize the work overload at the workstations. As future work, this algorithm will be integrated to the implemented algorithm to solve mixed-model ALBP.

5 Experimental Results

In this section, we present the experiments results. First, we present and compare the results for the mixed-model ALBP. Second, we analyze the impact of applying sequencing to minimize the work overload. The dataset was provided by manufactures partners and it reflects a real world scenario. It consists of real data and it is represented as a set of: 9 workstations, 9 models, 9 operators and 109 tasks. The MILP models were implemented using IBM ILOG CPLEX Optimization Studio, Version: 12.3. The heuristics were coded in C++.

Mixed-model assembly line balancing problem

Figure 2 represents the real case scenario. We are able to observe that some overload is registered, however this overload varies from model to model. For instance at workstation 7, model 2 produces a high under load meanwhile model 3 produces overload. This variation reflects the different operation times between models. The MILP model and heuristics proposed another distribution of tasks and consequently a different workstation load.

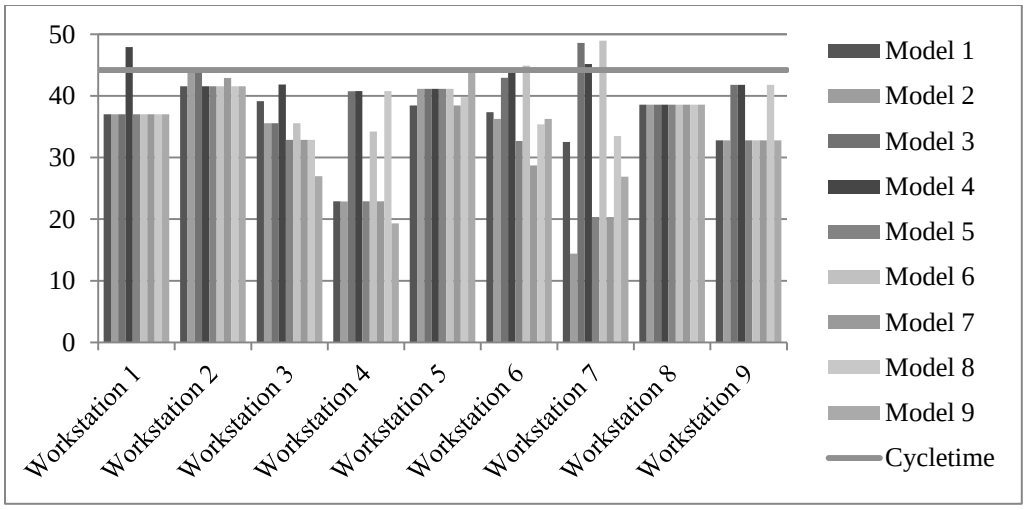


Figure 2 – Mixed-model assembly line balancing – Real case scenario

Figure 3 shows the results when applying the MILP model. The model is able to generate the optimal solution for the problem and it is used to analyze the efficiency of the heuristics procedures.

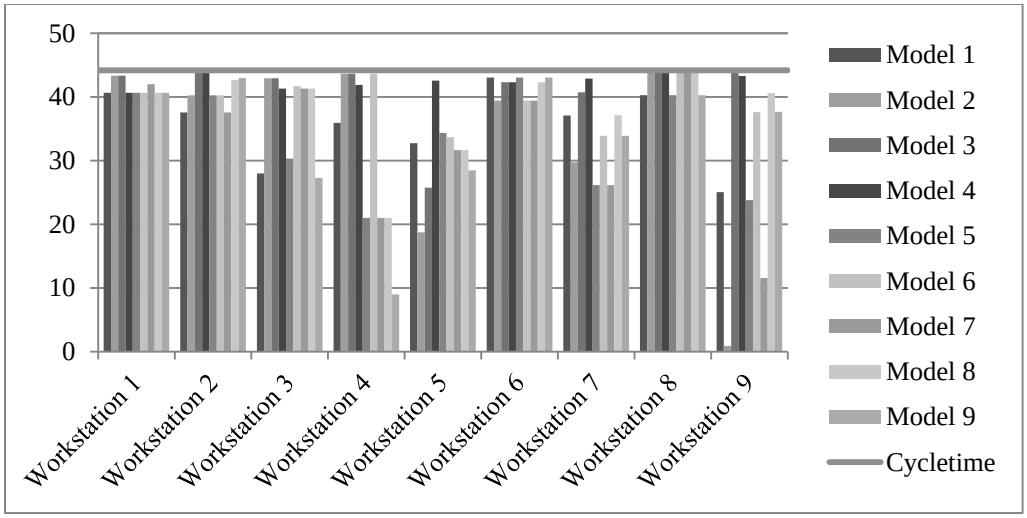


Figure 3 – Mixed-model assembly line balancing – MILP model

Figure 4 shows the results when applying the heuristics procedures. However the optimal solution is obtained for the super model, we are able to notice that when the proposed balance is applied to each model some overload is registered. Although it is not the optimal solution, the heuristics can be applied to bigger datasets and solve real case scenarios.

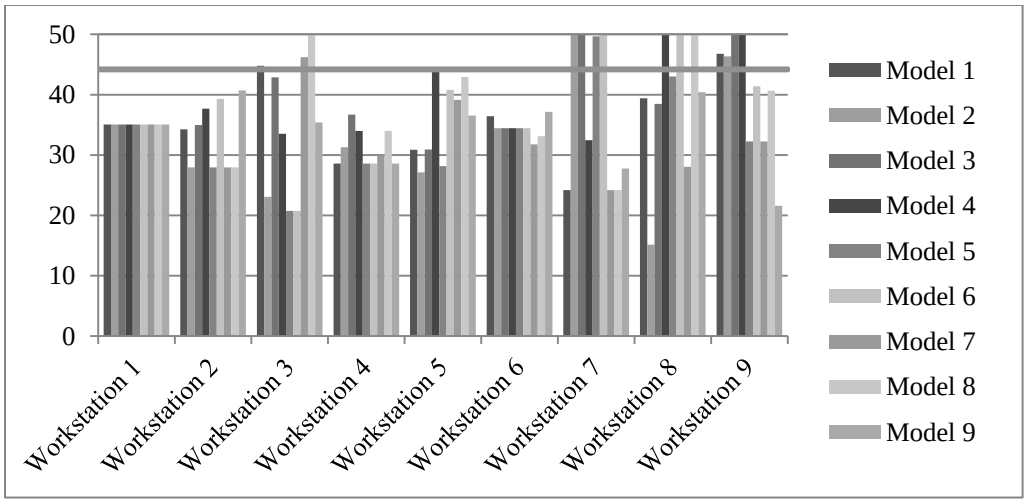


Figure 4 – Mixed-model assembly line balancing – Heuristics procedures

TABLE I represents the overall comparison results. It contains the average overload of each workstation, i.e. the average overload taking into account the 9 workstations. The last row emphasizes the total overload produced at each workstation. However the solutions to solve ALB problems focus on assigning tasks taking into consideration the overload, we can notice that some workstations still have some overload. The objective of the sequencing solution is to minimize this overload.

TABLE I – ALBP – Average workstations overload

	Real Case	MILP – ALBP	Heuristics
Workstation 1	0.40	0.00	0.00
Workstation 2	0.00	0.00	0.00
Workstation 3	0.00	0.00	1.12
Workstation 4	0.00	0.00	0.00
Workstation 5	0.00	0.00	0.01
Workstation 6	0.07	0.00	0.00
Workstation 7	1.09	0.00	6.64
Workstation 8	0.00	0.00	2.73
Workstation 9	0.00	0.00	5.33
Total Overload	14.1	0.00	142.42

Mixed-model assembly line sequencing problem

TABLE II represents the results obtained when applying the MILP model. The solution is very close to the optimal since almost all workstations have no overload. We are able to observe the efficiency of sequencing. The total workstations overload was drastically reduced in more than 85%.

TABLE II – Workstations overload – Overall results

	MILP – ALBP	Heuristics – ALBP	MILP –ALSP
Workstation 1	0.00	0.00	0
Workstation 2	0.00	0.00	0
Workstation 3	0.00	1.12	2.33

Workstation 4	0.00	0.00	0
Workstation 5	0.00	0.01	0
Workstation 6	0.00	0.00	0
Workstation 7	0.00	6.64	0
Workstation 8	0.00	2.73	0
Workstation 9	0.00	5.33	0
Total Overload	0.00	142.42	21

6 Conclusions and future work

This paper used different models and heuristics to solve mixed-model assembly line balancing and sequencing. The main objective was to minimize the work overload at assembly workstations. Initially, a MILP model and heuristics were developed to define the line balance. Using the results obtained by the balancing methods, a MILP model was coded and heuristics proposed. An experiment was carried out based on a set of 9 workstations, 9 models and 109 tasks. The MILP ALB model was able to generate the optimal solution. The ALBP heuristics was able to generate an optimal solution but only when considering the super model. The MILP ALS model was able to efficiently reduce the workload into 85%. Further research on the impact of sequencing to reduce overload in mixed-model assembly lines is currently on-going. The ALBP heuristics procedures are being improved and ALSP heuristics procedures implemented.

Acknowledgements The authors would like to thank the industrial collaborators: Ford Genk, SAS, SML Genk, VCST, Van Hool Industrial Vehicles, Volvo Cars Ghent and Volvo Trucks Ghent. The authors also gratefully acknowledge the support of Flanders' Drive, the innovation center for the vehicle industry in Flanders.

References

1. Valhondo, J. B. and Gude, J. P., Sequencing JIT Mixed Model Assembly Lines Under Station-Load and Part-Usage Constraints using Lagrangean Relaxations, MISTA - Multidisciplinary International Scheduling conference, Paris (2007).
2. Becker C. and Scholl, A., A survey on problems and methods in generalized assembly line balancing, European Journal of Operational Research, vol. 168, pp. 694-715 (2006).
3. Baybars I., A survey of exact algorithms for the simple assembly line balancing problem, Management science, vol. 32, pp. 909-932 (1986).
4. MacDuffie, J. P., Sethuraman, K. and Fisher, M. L., Product variety and manufacturing performance: evidence from the international automotive assembly plant study, Management Science, vol. 42, pp. 350-369 (1996).
5. Boysen, N., Kiel, M. and Scholl, A., Sequencing mixed-model assembly lines to minimise the number of work overload situations, International Journal of Production Research, vol. 49, pp. 4735-4760 (2011).
6. Boysen, N, Fliedner, M. and Scholl, A., Production planning of mixed-model assembly lines: Overview and extensions, Friedrich-Schiller-University Jena, School of Economics and Business Administration (2007).
7. Celano, G., Costa, A., Fichera,S. and Perrone, G., Human factor policy testing in the sequencing of manual mixed model assembly lines, Computers & Operations Research, vol. 31, pp. 39-59 (2004).
8. Xiaobo, Z and Ohno, K., Properties of a sequencing problem for a mixed model assembly line with conveyor stoppages, European Journal of Operational Research, vol. 124, pp. 560-570 (2000).
9. Xiaobo, Z and Ohno, K., Algorithms for sequencing mixed models on an assembly line in a JIT production system, Computers & Industrial Engineering, vol. 32, pp. 47-56 (1997).
10. Sumichrast, R. T., Oxenrider, K. A. and Clayton, E. R., An evolutionary algorithm for sequencing production on a paced assembly line, Decision Sciences, vol. 31, pp. 149-172 (2000).
11. Scholl, A., Klein, R. and Domschke, W., Pattern based vocabulary building for effectively sequencing mixed-model assembly lines, Journal of Heuristics, vol. 4, pp. 359-381 (1998).
12. Helgeson, W. B. and Birnie, D. P., Assembly Line Balancing using the Ranked Positional Weight technique, Journal of Industrial Enginering, vol. 12, pp. 394-8 (1961).

MISTA 2013

Minimizing the weighted sum of maximum earliness-tardiness cost on a single machine under maintenance constraints

Rachid Benmansour · Hamid Allaoui · Abdelhakim Artiba · Saïd Hanafi

1 Introduction

The importance of the just in time approach has led to a wide range investigation of scheduling problems that include both earliness and tardiness penalties [1]. In such a demanding environment, manufacturing systems must be in perfect conditions whenever needed. Many authors assume that the production system is available all the time. When unavailability periods are considered, there are few researchers that explicitly try to integrate preventive maintenance and scheduling decisions on a single machine. For example Graves and Lee [2] consider the problem to optimize weighted completion time and they take into consideration only one preventive maintenance period. Ji et al. [3] consider the same problem with the objective of minimizing the makespan. Diedrich et al. [20] studied the problem of minimizing the makespan on m identical parallel machines in the presence of reservations. For an arbitrary constant m they present a polynomial-time approximation scheme (PTAS) and show that no fully polynomial-time approximation scheme (FPTAS) exists unless P = NP holds, even if the number of reservations per machine is restricted to one. Furthermore, Eyraud-Dubois et al. [19] provide a comprehensive analysis for this problem for several variants of list scheduling algorithms and prove that the problem is inapproximable in the general case. Wang et al. [4] consider the problem of minimizing the total weighted job completion times on a single machine with availability constraints. They show that the problem is NP-hard in the strong sense and propose heuristics for the special case when the weights are

Rachid Benmansour
Université de Valenciennes et du Hainaut Cambrésis
E-mail: rbenmans@gmail.com

Hamid Allaoui
Université d'artois
E-mail: hamid.allaoui@univ-artois.fr

Abdelhakim Artiba
Université de Valenciennes et du Hainaut Cambrésis
E-mail: abdelhakim.artiba@univ-valenciennes.fr

Saïd Hanafi
Université de Valenciennes et du Hainaut Cambrésis
E-mail: said.hanafi@univ-valenciennes.fr

proportional and when there is only a single availability constraint. For more details, an excellent survey on scheduling with availability constraints can be found in the paper of Ma et al. [7]. In this survey, authors present recent main complexity results concerning the joint scheduling of production with unavailability periods in single machine, parallel machine, flow shop, open shop and job shop environment.
In this paper we consider the problem of minimizing the weighted sum of maximum earliness-tardiness cost on a single machine under maintenance constraints. As far as we know, this kind of objective function was only studied under the condition of equal unitary weights and without any consideration of maintenance constraints. Tavakkoli-Moghaddam et al. [9] consider the problem minimizing the maximum earliness and tardiness with general due dates. They use an idle insert algorithm and illustrate its efficiency by solving problems with different job sizes. They report also the optimal value associated with the special case of a common due date. In a subsequent paper Tavakkoli-Moghaddam et al. [10] present two methods to solve the above problem. Based on several experimental results, they show that the proposed simulated annealing method has a small error and a lower computational time than the branch-and-bound method. Recently, Moslehi et al. [11] consider the minimization of the sum of maximum earliness and tardiness in a flow shop configuration. They present optimal scheduling in a two-machine flow shop ($n/2/P/ET_{max}$) and determine dominant set for any optimal solution. They use a branch-and-bound method to solve the problem and introduce a number of lemmas to develop an effective algorithm which solve to optimality more than 82% of a selection of problems.

2 Mathematical models

We consider the following single machine scheduling problem, P1 : n independent jobs are simultaneously available for processing at time zero, and their processing times p_i for $i \in N = \{1, 2, ..., n\}$, are known and fixed in advance. All jobs have the same restrictive due date d and they are to be processed non-preemptively; furthermore the machine can handle only one job at any time. The objective is to jointly minimize the weighted sum of maximum earliness and maximum tardiness penalties

$$f = \alpha E_{max} + \beta T_{max} \tag{1}$$

where α, β, E_{max} and T_{max} are respectively the per unit time earliness penalty, the per unit time tardiness penalty, the maximum earliness and the maximum tardiness. We define furthermore C_i as the completion time of job $i \in N$ and $p_{max} = \max_{i \in N}\{p_i\}$ as the largest processing time of all jobs.If C_i is smaller than or equal to the common due date d, the job earliness is $E_i = d - C_i$. Accordingly, job i is tardy with the tardiness $T_i = C_i - d$, if its completion time is greater than the common due date d. As it is not known in advance whether a job will be completed before or after the due date, earliness and tardiness are calculated as $E_i = max\{d - C_i, 0\}$ and $T_i = max\{C_i - d, 0\}$ $\forall i \in N$. The maximum earliness E_{max} and the maximum tardiness T_{max} are defined as follows: $E_{max} = \max_{i \in N}\{E_i\}$ and $T_{max} = \max_{i \in N}\{T_i\}$.
First we study the problem without any availability consideration (problem P1). We present hereafter a mixed integer linear programming (MIP) formulation, that can be used to solve the problem to optimality.

2.1 Model 1

The following mixed integer linear programming (MIP) formulation, can be used to obtain optimal solution of the problem P1.

$$min\ f = \alpha E_{max} + \beta T_{max} \tag{2}$$
$$s.t.\ C_i \leq C_j - p_j + M(1 - x_{ij}) \quad \forall (i,j) \in N^2,\ j > i \tag{3}$$
$$C_j \leq C_i - p_i + M x_{ij} \quad \forall (i,j) \in N^2,\ j > i \tag{4}$$
$$T_i - E_i = C_i - d \quad \forall i \in N \tag{5}$$
$$C_i \geq p_i \qquad \forall i \in N \tag{6}$$
$$E_{max} \geq E_i \qquad \forall i \in N \tag{7}$$
$$T_{max} \geq T_i \qquad \forall i \in N \tag{8}$$
$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in N^2,\ j > i \tag{9}$$
$$T_i, E_i \geq 0 \qquad \forall i \in N. \tag{10}$$

In this model, the binary variable $x_{ij} = 1$ if the job i is sequenced before the job j, and 0 otherwise, the parameter M is sufficiently large scalar. The equation (2) represents the objective function to be minimized. Constraints of form (3) and (4) indicate the completion time of each job: if the variable $x_{ij} = 1$, the constraint (3) yields to $C_i \leq C_j - p_j$ and the constraint (4) becomes redundant. On the other hand, if $x_{ij} = 0$, the constraint (4) becomes $C_j \leq C_i - p_i$ and the constraint (3) becomes redundant. Note that the variables x_{ij} are defined only for $i < j$, as proposed by Manne [18] to define only $\frac{n(n-1)}{2}$ binary variables, rather than n^2.
The equation (5) is due to the fact that $E_i = \max(0, d - C_i) = d - \min(C_i, d)$ and $T_i = \max(0, C_i - d) = C_i - \min(C_i, d)$. It is used to calculate tardiness and earliness of each job. Constraint (6) assures that initial time of each job i is non negative. Constraints (7) and (8) compute the maximum earliness and the maximum tardiness respectively, while constraint (9) defines the variable x_{ij} as binary. Finally restrictions (10) defines the non negativity of variables T_i and E_i.
We use ILOG CPLEX 11.2 to solve this model. Numerical experiments show that computing time is large even for small instances ($n = 10$). This is due in particular we were looking to determine the position of each job in the optimal schedule. We will show in the following that only the first job (and its starting time) should be determined regardless of the position of the other jobs. Furthermore we show that there is no idle time in the optimal schedule. In view of these considerations, we propose a linear model that solves the problem to optimality and secondarily we show that the optimal schedule depends only on the values of α, β, d and the largest processing time p_{max}.

2.2 Model 2

Since we are dealing with a restrictive problem, the last job in every feasible schedule will complete necessarily after the due date d. Therefore, the maximum tardiness T_{max} is always strictly positive. Below we give some properties of the problem P1, then we propose a mathematical model to solve it to optimality.

Propriety 1 *In an optimal schedule there are no idle times between two consecutive jobs.*

Proof Suppose in an optimal schedule σ, we have an idle time of duration δ between two consecutive jobs i and j. If these jobs are early then we can improve the value of f by moving i and all the jobs before it by δ towards the due date. In this case the new maximum earliness will decrease by $\alpha\delta$. The same reasoning can be conducted if the two jobs are tardy, in this case we can improve the tardiness cost by scheduling job j immediately after the completion of i. Finally if the idle time coincides with d, the objective function is improved by scheduling the first tardy job j at time d and the last early job i at $d - p_i$.

Propriety 2 *In an optimal schedule, the starting time of the job processed first is less or equal than d.*

Proof Suppose S^* is an optimal schedule in which the first job starts at time $t > d$. In this case the value of the objective function is $f(S^*) = \beta\left((t-d) + \sum_{i=1}^{n} p_i\right)$. It is clear that this value can be improved by starting the first job exactly at time d. This leads to a contradiction with the fact that S^* is an optimal schedule. Hence the starting time of first job in the schedule is less or equal than d.

Propriety 3 *In an optimal schedule, the order of jobs is not important as long as the first job and its starting time are determined.*

Proof Since there is no idle time between two consecutive jobs, the maximum tardiness value depends only the starting time t_0 of the schedule: $T_{max} = t_0 + P - d$ (where $P = \sum_i p_i$). On the other hand, the maximum earliness depends on t_0 and on the processing time p_i of the first job scheduled: $E_{max} = \max(0, d - t_0 - p_i)$.

Based on these proprieties, we propose a second model that determines the starting time of the schedule and the first job in an optimal schedule. We define some additional variables according to figure 1. We denote by I the idle time before the processing of the first job and by Y the amount of work of the first job completed before d.
Model 2 is as follows:

$$min\ f = \alpha E_{max} + \beta T_{max} \tag{11}$$
$$s.t.\ I + Y + E_{max} = d \tag{12}$$
$$Y + E_{max} + T_{max} = P \tag{13}$$
$$Y \leq max(d, p_{max}) \tag{14}$$
$$E_{max} \leq MY \tag{15}$$
$$I,\ Y \geq 0 \tag{16}$$
$$E_{max},\ T_{max} \geq 0. \tag{17}$$

The constraint (12) indicates that the first job starting processing before d, along with the maximum earliness, does not exceed the duration d. The remaining time, if any, determines the value of variable I. Constraint (13) simply reflects the fact stated in property (1). Constraint (14) requires that the variable Y does not exceed neither the value of d nor the value of p_{max}. The next constraint (16) requires the variable E_{max} to take the value 0 whenever $Y = 0$. If $Y > 0$ then constraint (15) is not restrictive for the variable E_{max}. The remaining constraint (16)-(17) defines the non negativity of variables I, Y, E_{max} and T_{max}.
This model solves to optimality problems with large instances instantaneously ($n =$

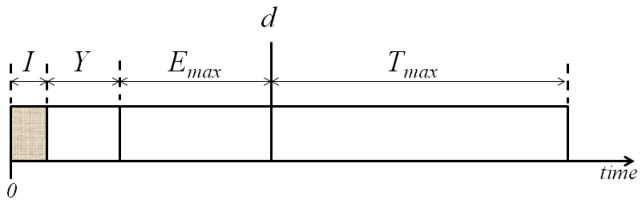


Fig. 1 A feasible schedule of Problem 1

1000). Furthermore, we derive some interesting formulas that allow us to determine the optimal value of the objective function f^*, the starting time of the schedule and the first job to be processed. Hence there is no need to use the models presented earlier once we know the value of α, β, d and p_{max}.

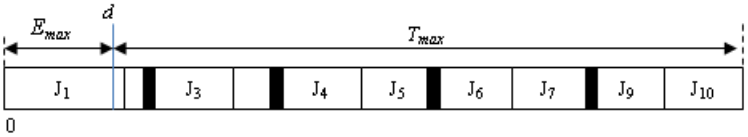


Fig. 2 Example of an optimal schedule for the $n = 10$ problem instance obtained by the MIP

2.3 Model 3

We suppose now, given the same assumptions of the Problem $P1$, that maintenance is required after a periodic time interval T. Of course the maintenance activity can occur before that time since we are dealing with a nonpreemptive scheduling problem. We suppose furthermore that each maintenance requires m units of time. The figure 3 show a feasible schedule with periodic preventive maintenance. In an optimal schedule, there will be several sets of jobs (batches) and after the completion of the last job in each batch, a preventive maintenance is conducted on the machine except for the last batch. The MIP hereafter solves this problem, denoted Problem $P2$, to optimality.

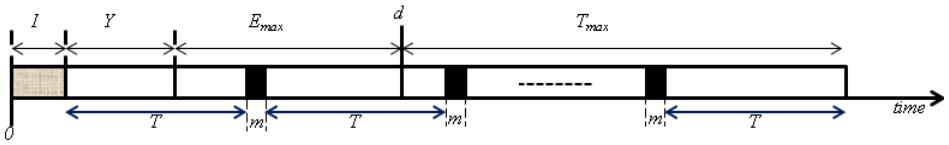


Fig. 3 A feasible schedule of Problem 2

$$min\ f = \alpha E_{max} + \beta T_{max} \tag{18}$$

$$s.t.\ I + Y + E_{max} = d \tag{19}$$

$$Y + E_{max} + T_{max} = P + m\left(\sum_{j=1}^{n} y_j - 1\right) \tag{20}$$

$$\sum_{i=1}^{n} p_i z_{ij} \leq T y_j \quad \forall j \in N \tag{21}$$

$$\sum_{j=1}^{n} z_{ij} = 1 \quad \forall i \in N \tag{22}$$

$$Y \leq d \tag{23}$$

$$Y \leq p_{max} \tag{24}$$

$$E_{max} \leq MY \tag{25}$$

$$I,\ Y,\ E_{max},\ T_{max} \geq 0 \tag{26}$$

$$z_{ij} \in \{0,1\} \quad \forall (i,j) \in N^2 \tag{27}$$

$$y_j \in \{0,1\} \quad \forall j \in N. \tag{28}$$

$$\tag{29}$$

Constraints (19) and (23)-(26) have already been described in model 2. The second restriction (20) imposes that the sum of all processing times and the maintenance durations is equal to the sum of the maximum earliness, the maximum tardiness and the processing time of the first scheduled job (or the part of this job that is before d). The term -1 in this equality is due to the assumption that no maintenance is needed after the completion of the last job; in this way, the optimal number of preventive maintenance $k = \sum_{j=1}^{n} y_j - 1$. In addition, constraint (21) states that no set of jobs processed together shall exceed the duration T (of course $T \geq p_i\ \forall i \in N$). In an optimal schedule, every job is assigned to one specific batch; this will be guaranteed by the constraint (22). Here the binary variable $z_{ij} = 1$ if i is assigned to batch j, and 0 elsewhere. The last two restrictions (27) and (28) defines the variables z_{ij} and y_j as binaries. It turns out that this problem depends only on the starting time of the first job, the duration of its processing time, and the minimal number of sets needed to form a feasible schedule. We show that this problem reduces to the one dimensional bin packing (NP-hard). It is worth mentioning that considering maintenance constraints is a valid assumption from an industrial point of view, since every machine is subject to random failure due to environmental conditions or aging. To solve the current problem, we use best known heuristics for the 1d bin-packing problem to solve large instances of the current problem.

3 Conclusion

It has been shown through the properties discussed in section 2 that the problem without maintenance is very easy to solve. The problem with maintenance, (which is NP-Hard) can be solved approximately using the First-Fit-Decreasing algorithm or

the Next-Fit-Decreasing algorithm. Once the minimal number of batches (or bins) is determined, we can compute the best schedule accordingly with no guarantee that this schedule is the best one. For small instances, the problem can be solved within few seconds using the proposed MIP. In future research, we intend to take into account the stochastic aspects of the presented problems; especially we will study the problem when the machine is subject to random failure. Another way to extend this study is to consider different maintenance strategies.

Acknowledgements The present research work has been supported by International Campus on Safety and Intermodality in Transportation the Nord-Pas-de-Calais Region, the European Community, the Regional Delegation for Research and Technology, the Ministry of Higher Education and Research, and the National Center for Scientific Research. The authors gratefully acknowledge the support of these institutions.

References

1. M. Pinedo, *Scheduling: theory, algorithms, and systems*, Springer, 2008.
2. G. Graves, C. Lee, Scheduling maintenance and semiresumable jobs on a single machine, *Naval Research Logistics* (NRL), 46, pp 845-863 (1999).
3. M. Ji, Y. He, T. Cheng, Single-machine scheduling with periodic maintenance to minimize makespan, *Computers & operations research*, 34, pp 1764-1770 (2007).
4. G. Wang, H. Sun, C. Chu, Preemptive scheduling with availability constraints to minimize total weighted completion times, *Annals of Operations Research*, 133, pp 183-192 (2005).
5. I. Kacem, C. Chu, A. Souissi, Single-machine scheduling with an availability constraint to minimize the weighted sum of the completion times, *Computers & operations research*, 35, pp 827-844 (2008).
6. C. Low, M. Ji, C. Hsu, C. Su, Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance, *Applied Mathematical Modelling*, 34, pp 334-342 (2010).
7. Y. Ma, C. Chu, C. Zuo, A survey of scheduling with deterministic machine availability constraints, *Computers & Industrial Engineering*, 58, pp 199-211 (2010).
8. R. Tavakkoli-Moghaddam, M. Vasei, A single machine sequencing problem with idle insert: Simulated annealing and branch-and-bound methods, *International Journal of Industrial Engineering & Production Research* , 19, pp 47–56 (2008).
9. R. Tavakkoli-Moghaddam, G. Moslehi, M. Vasei, A. Azaron, Optimal scheduling for a single machine to minimize the sum of maximum earliness and tardiness considering idle insert, *Applied mathematics and computation*, 167, pp 1430-1450 (2005).
10. R. Tavakkoli-Moghaddam, G. Moslehi, M. Vasei, A. Azaron, A branch-and-bound algorithm for a single machine sequencing to minimize the sum of maximum earliness and tardiness with idle insert, *Applied mathematics and computation*, 174, pp 388-408 (2006).
11. G. Moslehi, M. Mirzaee, M. Vasei, M. Modarres, A. Azaron, Two-machine flow shop scheduling to minimize the sum of maximum earliness and tardiness, *International Journal of Production Economics*, 122, pp 763-773 (2009).
12. D. Biskup, M. Feldmann, Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates, *Computers & Operations Research*, 28, pp 787-801 (2001).
13. K. Baker, G. Scudder, Sequencing with earliness and tardiness penalties: a review, *Operations research*, 38, pp 22-36 (1990).
14. R. Graham, E. Lawler, J. Lenstra, A. Kan, Optimization and Approximation in Deterministic Sequencing Scheduling: a Survey, *Annals of Discrete Mathematics*, 5, pp 287-326 (1979).
15. A. Keha, K. Khowala, J. Fowler, Mixed integer programming formulations for single machine scheduling problems, *Computers & Industrial Engineering*, 56, pp 357-367 (2009).
16. D. Ronconi, M. Kawamura, The single machine earliness and tardiness scheduling problem: lower bounds and a branch-and-bound algorithm, *Computational & Applied Mathematics*, 29, pp 107-124 (2010).

17. C. Hsu, C. Low, C. Su, A single-machine scheduling problem with maintenance activities to minimize makespan, *Applied Mathematics and Computation*, 215, pp 3929-3935, (2010).
18. A. S. Manne, On the job-shop scheduling problem, *Operations Research*, 8, pp 219–223 (1960).
19. . L. Eyraud-Dubois, M. Grégory, D. Trystram, Analysis of scheduling algorithms with reservations, *Parallel and Distributed Processing Symposium, IPDPS 2007. IEEE International*, pp 1–8 (2007).
20. . F. Diedrich, K. Jansen, F. Pascual, D. Trystram, Approximation algorithms for scheduling with reservations, *Algorithmica, Springer*, 58(2), pp 391-404 (2010).

MISTA 2013

Real-life Examination Timetabling

Tomáš Müller

Abstract An examination timetabling problem at a large American university is presented. Though there are some important differences, the solution approach is based on the ITC 2007 winning solver which is integrated in the open source university timetabling system UniTime. In this work, nine real world benchmark data sets are made publicly available and the results on four of them are presented in this paper. A new approach to further decreasing the number of student conflicts by allowing some exams to be split into multiple examination periods is also studied.

1 Introduction

Examination timetabling is a well researched and important educational timetabling problem (Qu et al, 2009). It consists of individual meetings (*examinations*) that are to be assigned in time and space in a way that allows all the participants (students and instructors) to attend all the meetings that they require. The time is usually split into a set of non overlapping time slots (*examination periods*) of equal or varying length, and each examination needs to be assigned into one period that is of the same length or longer than the exam. Similarly, an examination needs to take place in a room that is big enough and there is usually only one examination in a room during one examination period. Besides direct conflicts (when a student or instructor is required to attend two examinations that are timetabled in same examination period) that need to be avoided or minimized, there are various other optimization criteria that need to be considered. For instance, it is often desired to avoid cases when a student needs to take an exam during two consecutive examination periods or having too many examinations during one day.

In this paper, we discuss a particular examination problem that is quite common at large American universities and that has been implemented in UniTime[1]. UniTime is a comprehensive university timetabling application that covers both post-enrollment (Rudová et al, 2011) and curriculum-based (Müller and Rudová, 2012)

Tomáš Müller
Space Management and Academic Scheduling, Purdue University
501 Northwestern Avenue, West Lafayette, USA
E-mail: muller@unitime.org

[1] `http://www.unitime.org`

course timetabling, examination timetabling, student scheduling (Müller and Murray, 2010), and event management. Also, nine real world instances from Purdue University are introduced in this work and made publicly available for benchmarking. This set covers almost 5 years of experience using UniTime for examination timetabling at the university (initially used in Fall 2008).

The examination problem at Purdue University is quite large. It consists of about 2,000 examinations, attended by up to 34,000 students with 120,000 student enrollments, placed in 29 distinct examination periods and using about 350 rooms. It is also quite hard to solve, as it is often impossible to find a complete timetable that has no direct conflicts. There are also a few, but substantial, differences to what has been studied so far (McCollum et al, 2012). For instance, multiple examinations in one room during the same period are not allowed, however, some of the large examinations require being split between multiple (usually up to 4) rooms. Also, two types of room seating are considered. An examination may either require a normal (class-like) seating or a special examination seating arrangement. This means that each room has two capacities: normal seating (usually equal to the number of chairs in a room) and examination seating (usually somewhere around half of the normal seating) which requires no two students sitting next to each other. Besides minimizing the number of direct conflicts, we also care about cases when a student has three or more examinations on a day, and about cases when two exams are back-to-back (a student must attend two exams that are on the same day in two consecutive periods). The more than two examinations on a day conflict is considered almost as bad as a direct conflict since in both cases a student may request an individual examination. In addition to these three types of student conflicts, there are several additional constraints and optimization criteria, like room availability and individual room and period preferences for an examination, that need to be considered.

It is worth mentioning that the largest problem (named PUR93) in Carter's famous and widely used benchmark data set (Carter et al, 1996) is based on older data from Purdue University. A solution to this problem using Carter's problem representation has never been used in practice at Purdue however.

The examination timetabling problem in UniTime is solved by an algorithm that is an extension of the winning algorithm in the second International Timetabling Competition[2] (ITC2007) using our Constraint Solver Library (Müller, 2009). It uses a constraint-based framework incorporating a series of algorithms based on local search techniques that operate over feasible, though not necessarily complete, solutions. In these solutions, some variables may be left unassigned, however, all hard constraints on assigned variables must be satisfied. The library is written in Java and is publicly available[3] under GNU's LGPL license.

The examination timetabling problem that has been studied in this work is described in the following section. The algorithm used to solve this problem is discussed next. The paper is concluded by a series of experiments on the four data sets collected at Purdue during the last four semesters.

[2] http://www.cs.qub.ac.uk/itc2007

[3] http://cpsolver.googlecode.com

2 Examination Timetabling Problem

The examination timetabling problem consists of a set of non-overlapping examination periods, a set of rooms, a set of examinations, and a set of distribution constraints.

Each **examination period** has defined a date, a start time and a length. It may also have a penalty associated with it. These penalties can be overridden on individual exams and are used to penalize certain unpopular examination periods (e.g., late evening or Saturday periods).

Each **room** has a normal and an examination seating capacity defined. A room can have GPS coordinates and there is a room availability matrix, stating during which periods a room can be used. Similarly as with examination periods, there can be a penalty associated with using the room at a certain period. Room availability and penalties can be expressed in a single two dimensional array containing infinite penalty for cases when a room is not available for examination timetabling at a certain period and an integer penalty otherwise. Note that we allow negative penalties which can be used to prefer a certain room or period over the others.

$$penalty_{rp}(r,p) = \begin{cases} \infty & \text{room } r \text{ not available at period } p \\ x \in \mathbb{Z} & \text{room } r \text{ has a penalty of } x \text{ at period } p \end{cases}$$

Each **examination** has defined a length, type of seating, maximal number of rooms into which it can be split (typically between 1 to 4), and a list of students enrolled in the exam. Additionally, each exam has a list of available periods and rooms associated with it. There can also be a penalty associated with each period or room of the exam for using the given period or room respectively.

All penalties in UniTime are based on the following scale. The room penalty on an examination is then computed by combining all such penalties set on individual rooms, buildings, room features and groups of rooms. For instance, there can be a penalty of -3 for a room with strongly preferred equipment that is located in a discouraged building.

-4 strongly preferred
-1 preferred
0 neutral / no penalty
1 discouraged
4 strongly discouraged
∞ prohibited

A **distribution constraint** is set between two or more exams and it can be either hard or soft. Hard constraints must be satisfied, soft constraints have a penalty that is incurred when the constraint is violated. Distribution constraints are of the following types:

- **Same Room:** Exams are required/expected to take place in the same room or rooms.
- **Different Room:** Exams are required/expected to take place in different rooms (no two exams of the constraint are to be placed in the same room).
- **Same Period:** Exams are required/expected to take place during the same periods.
- **Different Period:** Exams are required/expected to take place during different periods.

– **Precedence:** Exams are required/expected to take place in the order they are listed in the constraint. For instance if exam A is in the precedence constraint before exam B, it is to be placed in a period that precedes the period in which exam B is placed.

For more details, please see the XML format[4] that is used to store the input data as well as the resultant solution.

2.1 Hard Constraints

The following hard constraints need to be respected while assigning exams to examination periods and rooms:

– **One At A Time:** Only one exam can be placed in a room at any period.
– **Room Availability:** A room cannot be used during periods for which it is not available. This means that no exam can be placed in a room r during a period p when $penalty_{rp}(r, p) = \infty$.
– **Exam Availability:** An exam cannot be placed in a period or a room that is not available for the exam. This means that an exam e can not be placed in a period where $penalty_{ep}(e, p) = \infty$ or into a room r where $penalty_{er}(e, r) = \infty$.
– **Exam Size:** An exam must be placed in a room (or set of rooms) so that the overall seating capacity of the rooms equals or is greater than the number of students attending the exam, with respect to the requested seating type. This means that an examination e can be assigned in rooms $r_1, r_2, ..., r_n$ only when

$$\sum_{i=1,2,...,n} size_{seating(e)}(r_i) \geq |students(e)|$$

where $size_{seating(e)}(r_i)$ is the capacity of room r_i of the seating requested by the examination e (denoted $seating(e)$) and $students(e)$ is the list of students enrolled in the exam e.
– **Max Rooms:** Maximum number of rooms into which an exam can be split cannot be exceeded.
– **Hard Distributions:** Hard distribution constraints must be satisfied.

Exams that do require to take place in the same room must be placed in different examination periods as the **Same Room** constraint does not override the **One At A Time** hard constraint. Similarly, exams that are to be placed in the same period can not be placed in the same rooms.

2.2 Soft Constraints

During the search, besides looking for a complete solution (all exams are assigned to periods and rooms) that satisfies all hard constraints mentioned above, the following criteria are optimized. Each criterion has a weight associated with it (e.g., direct conflicts have typically much higher weight that back-to-back conflicts), the overall weighted sum of the following criteria is minimized.

[4] `http://www.unitime.org/exam_dataformat.php`

- **Direct Conflict:** A student is enrolled in two exams that are placed in the same period. If a student is attending three exams that are placed in the same period, it is counted as three direct conflicts (there is a direct conflict for every pair of two exams).
- **More Than Two Exams A Day Conflict:** A student is enrolled into three or more exams that take place on the same day.
- **Back-To-Back Conflict:** A student is enrolled into exams that are scheduled in consecutive periods (back-to-backs over an end of a day are not considered).
- **Period Penalty:** A penalization of an assignment of an exam into a period. It is a combination of the penalty $penalty_{ep}(e,p)$ set on the exam e for a particular period p and a default penalty that is set on the period itself $penalty_p(p)$.

$$PeriodPenalty(e,p) = 2 \times penalty_{ep}(e,p) + penalty_p(p)$$

  The constant 2 in the equation is used to give the individual examination penalty greater weight than the default penalty that is set on the period and applies to all examinations assigned in that period.
- **Room Penalty:** A penalization of an assignment of an exam into a room. It is a combination of the penalty $penalty_{er}(e,r_i)$ set on the examination e for a particular room r_i and the penalty $penalty_{rp}(r_i,p)$ set on the room r_i for the period p that is assigned. If an examination is assigned in two or more rooms $(r_1, r_2, ..., r_n)$, the room penalty for the exam is a sum of the room penalties for each of the assigned rooms.

$$RoomPenalty(e,p,r_1,r_2,...,r_n) = \sum_{i=1,2,...,n} (2 \times penalty_{er}(e,r_i) + penalty_{rp}(r_i,p))$$

- **Distribution Penalty:** A penalty for a soft distribution constraint that is not satisfied. It is usually either 1 for preferred constraints or 4 for strongly preferred constraints.
- **Room Split:** A penalization for assigning an exam into multiple rooms. There is penalty of 1 of splitting an exam into two rooms, 4 into three rooms, 9 into four rooms. In general, it is $(n-1)^2$, where n is the number of rooms into which an exam is assigned.

$$RoomSplit(e,r_1,r_2,...,r_n) = \begin{cases} 0 & n \leq 1 \\ (n-1)^2 & n > 1 \end{cases}$$

- **Room Split Distance:** A penalty for splitting an exam into rooms that are not close to each other. If an exam is split into two or more rooms, it is the average direct distance between these rooms. Distances are computed in meters between rooms that have GPS coordinates provided.

$$RoomSplitDistance(e,r_1,r_2,...,r_n) = \frac{2}{n(n-1)} \sum_{i,j \in \{1,2,...,n\}; i<j} distance(r_i,r_j)$$

- **Room Size:** A penalty for using rooms that are too large for an exam. It is the difference between the size of the room (either exam or normal seating based on what is required by the exam) and the number of students enrolled in the exam. If an exam is assigned into two or more rooms, it is the difference between the size of the exam and the total sum of the room sizes in the seating type of the exam.

Between two exams, if it is desired to have a smaller exam in the smaller room (assuming that both rooms are big enough for both exams), the **Room Size** penalty can be raised to the power of some additional factor. For instance, with the room size factor $f = 1.1$, having an exam of 30 students in a room of 100 would create a penalization of $(100 - 30)^{1.1} = 107.06$, instead of just 70.

$$RoomSize(e, r_1, r_2, ..., r_n) = (\sum_{i=1,2,...,n} size_{seating(e)}(r_i) - |students(e)|)^f$$

- **Rotation Penalty:** An exam can have an average period placement (computed from previous examination problems) associated with it. If so, a penalty equal to the square root of the multiplication of the current period index (i.e., 1, 2, ..., 29) and the average period placement is associated with each exam. This effectively "rotates" the exam time placements by increasing the penalty on solutions in which exams with historically late period assignments are placed in late periods again.

$$RotationPenalty(e, p) = \sqrt{index(p) \times average(e)}$$

The average period placement $average(e)$ of an exam e is computed as the average of
  - the period index of the exam during the previous semester $index_p(e)$ and
  - the average period placement value the exam had at that time $average_p(e)$.

The period index of the exam during the previous semester is taken if the exam did not have the average period placement set at that time.

$$average(e) = \begin{cases} 0 & e \text{ is a new exam} \\ index_p(e) & e \text{ was new in previous term} \\ \frac{1}{2}index_p(e) + \frac{1}{2}average_p(e) & \text{otherwise} \end{cases}$$

For example, if an exam is going to be timetabled in Fall 2012 for the fifth time, the average period placement for the Fall 2012 examination timetabling problem is computed from the assignment of the previous four Fall[5] timetabling problems (2008, 2009, 2010, and 2011) as follows:

$$\begin{aligned} average_{12}(e) &= {}^{index_{11}(e)}/_2 + {}^{average_{11}(e)}/_2 \\ &= {}^{index_{11}(e)}/_2 + {}^{index_{10}(e)}/_4 + {}^{average_{10}(e)}/_4 \\ &= {}^{index_{11}(e)}/_2 + {}^{index_{10}(e)}/_4 + {}^{index_{09}(e)}/_8 + {}^{index_{08}(e)}/_8 \end{aligned}$$

This gives the more recent semesters more weight while keeping the older assignments in mind as well.

- **Large Exams Penalty:** One for each large exam (an exam with more than a certain number of students enrolled) that is placed on or after a certain period
- **Room Distance:** If an examination was created for a particular class (as opposed to the whole course), it is often desired for the exam to take place in the same room where the class took place (called *original room*). This is modeled by having a negative penalty of -4 on such a room and a penalty of -1 on all the other available rooms of the same building as the original room. If other than the original room is assigned to the exam, the **Room Distance** criterion measures the distance between the assigned room and the original room. If the exam is in multiple rooms, the average distance between the assigned rooms and the original room of the class is used as a penalty for not having the original room assigned.

[5] Since there are typically different courses offered in Spring and Fall, we usually consider only semesters of the same season for the average periods.

2.3 ITC 2007

Following are the main differences between the examination timetabling problem described in this paper and the examination problem of the first track of the second International Timetabling Competition (ITC 2007). See (Müller, 2009) for more details about the application of the described algorithm on the competition instances.

- Direct student conflicts are allowed, but minimized. In the competition problem it is not allowed for a student to have two exams at the same period.
- We do not allow two exams to share a room. On the other hand, an exam may be split among multiple rooms.
- Rooms have two capacities, based on the seating (examination or normal seating).
- The presented problem allows for room and period restrictions and penalties that can be set for individual exams. The room and period penalties of the competition problem applies to all exams using the given room and/or period.
- Besides of direct and back-to-back student conflicts which are present in both problems, instead of the more than two exams on a day, the competition problem has two exams on a day and a more general period spread. Period spread gives a penalty when a student has two exams over a specified number of consecutive periods.

While there is no direct mapping between the two problems, there are a lot of similarities as well. For the presented algorithm, the competition problems appear to be harder in finding a complete solution. This is mostly because direct student conflicts are not allowed. But there also seem to be less space available, which also corresponds with the inability to split an exam into multiple rooms if needed and the general need for some exams to share a room. On the other hand, the problems presented in this work are considerably larger, in many cases with fewer examination periods, and the exams seem to be more interlinked with students. As is shown in the experiments, while disallowing direct student conflicts is certainly possible, it results in a dramatic increase of back-to-back and more than two exams a day conflicts for students.

3 Algorithm

The search algorithm consists of several phases: In the first (construction) phase, a complete solution is found using an Iterative Forward Search (IFS) algorithm (Müller, 2005). This algorithm makes use of Conflict-based Statistics (CBS) (Müller et al, 2004) to prevent itself from cycling. In the next phase, a local optimum is found using a Hill Climbing (HC) algorithm. Once a solution can no longer be improved using this method, the Great Deluge (GD) technique (Dueck, 1993) is used. The GD algorithm is altered so that it allows some oscillations of the bound that is imposed on the overall solution value. The search ends after a predetermined time limit has been reached. The best solution found within that limit is returned.

3.1 Construction Phase

Initially, a complete solution is found using the Iterative Forward Search algorithm (see Figure 1). It starts with all variables (examinations) being unassigned (Line 2). During each iteration, an unassigned variable is selected (Line 4) and a value from its

domain is assigned to it (assignment of an examination period and a room or rooms). If this causes any violations of hard constraints with existing assignments, the conflicting variables are unassigned (Lines 5 - 7). For example, if there is another examination in the selected room at the selected period, that exam will be unassigned. More precisely, the function HARDCONFLICTS (Line 6) returns a subset γ of the solution σ, so that there are no hard constraints violated between the remaining assignments $\sigma\backslash\gamma$ and the new assignment $\{v/d\}$ of the selected variable v (Line 4) with the selected value d (Line 5). The search ends when all exams are assigned or a time limit is reached (Line 3).

1: **function** CONSTRUCTION()
2: $\sigma = \emptyset$
3: **while not** COMPLETE(σ) **and not** TIMEOUT **do**
4: $v =$ SELECTVARIABLE(σ)
5: $d =$ SELECTVALUE(σ, v)
6: $\gamma =$ HARDCONFLICTS(σ, v, d)
7: $\sigma = \sigma\backslash\gamma \cup \{v/d\}$
8: **return** σ

Fig. 1 Pseudo-code of the construction algorithm.

The search is also parametrized by variable and value selection criteria (Lines 4 and 5). The examinations are ordered based on the following criteria

- *domain size:* examinations with the smallest domain are assigned first
- *highest degree:* among exams with the same domain size, examinations with the highest number of other exams that have at least one student in common with the exam are assigned first
- *examination size:* among examinations with the same domain size and the number of correlated exams, the largest examinations (in the number of students taking the exam) are assigned first

If there are two or more unassigned exams with the same domain size, number of correlated exams and students, one is chosen randomly. For the selected variable, a value whose assignment increases the overall cost of the solution the least is selected among values that violate the smallest number of hard constraints (i.e., the number of conflicting variables that need to be unassigned in order to make the problem feasible after assignment of the selected value to the selected variable is minimized). If there is a tie, one of these is selected randomly. It does not go through the whole domain, but the best possible room placement is selected for each of the available periods. If there is no room placement that does not violate any hard constraint (e.g., all the big enough rooms are already occupied by some other exams), a valid room assignment is selected randomly.

Conflict-based Statistics is used during this process to prevent repetitive assignments of the same values by memorizing conflicting assignments. Conflict-based Statistics is a data structure that memorizes hard conflicts which have occurred during the search together with their frequency and the assignments that caused them. More precisely, it is an array

$$CBS[V_a = v_a \ \rightarrow \ \neg\, V_b = v_b] = c_{ab}.$$

This means that the assignment $V_a = v_a$ has caused a hard conflict c_{ab} times in the past with the assignment $V_b = v_b$. Note that this does not imply that the assignments

$V_a = v_a$ and $V_b = v_b$ cannot be used together in the case of non-binary constraints. In the value selection criterion, each hard conflict is then weighted by its frequency, i.e., by the number of past unassignments of the current value of the conflicting variable caused by the selected assignment (see Figure 2).

```
1: function selectValue(σ, v)
2:  α = ∅, p_min = ∞, h_min = ∞
3:  for a in valuesToCheck(v) do
4:      γ = hardConflicts(σ, v, a)
5:      h = Σ_{w/b∈γ} (1 + CBS[v, a, w, b])
5:      p = penalty(σ\γ ∪ {v/a})
6:      if h < h_min or ( h = h_min and p ≤ p_min ) then
7:          if h < h_min or ( h = h_min and p < p_min ) then α = ∅, h_min = h, p_min = p
8:          α = α ∪ a
9:  a = random(α)
10: for w/b in hardConflicts(σ, v, a) do
11:     CBS[v, a, w, b] = 1 + CBS[v, a, w, b]
12: return a
```

Fig. 2 Pseudo-code of the value selection criterion using the Conflict-based Statistics.

If a complete solution is not found during the construction phase, it is very likely due to an inconsistency in the input data. In such a case, the content of the Conflict-based Statistics provides very valuable information back to the user which can be used to identify the problematic hard constraint or constraints that need to be relaxed.

3.2 Hill Climbing Phase

Once a complete solution is found, a Hill Climbing algorithm is used in order to find a local optimum (see Figure 3). In each iteration a change in the assignment of the current solution is proposed by random selection from a problem-specific neighborhood (Lines 4 - 5). The generated move is only accepted when it does not worsen the overall solution value (i.e., the weighted sum of violated soft constraints, Lines 6 - 8). Only changes that do not violate any hard constraints are considered. This rule applies to the following phase as well. Neighbor assignments are also generated consistently throughout all phases. That is, a problem specific neighborhood is selected randomly and is used to generate a random change in the current solution.

```
1: function hc(σ)
2:  S_best = penaly(σ), idle = 0
3:  while idle ≤ HC_idle and not timeout do
4:      n = random({PeriodChange, RoomChange, ExamSwap, RandomMove})
5:      δ = generate(σ, n),   S = penalty(σ ⊗ δ)
6:      if S ≤ S_best then
7:          σ = σ ⊗ δ
8:          if S < S_best then idle = 0, S_best = S
9:      idle = idle + 1
10: return σ
```

Fig. 3 Pseudo-code of the Hill Climbing algorithm.

The hill climbing phase is finished after a specified number HC_{idle} of idle iterations during which a solution has not improved (counted in variable *idle* on Figure 3). In all the following experiments, the parameter HC_{idle} was set to $25,000$.

3.3 Great Deluge Phase

The Great Deluge algorithm uses a bound B that is imposed on the overall value of the current solution that the algorithm is working with (see Figure 4). This means that the generated change is only accepted when the value of the solution after an assignment does not exceed the bound (Lines 6 - 8). The bound starts at the value

$$B = GD_{ub} \cdot S_{best}$$

where S_{best} is the overall value of the best solution found so far (Line 1), and GD_{ub} is a problem specific parameter (upper bound coefficient, set to 1.05 in all the experiments). The bound is decreased after each iteration (Line 9). This is done by multiplying the bound by a cooling rate GD_{cr} (set to 0.99999995 in all the experiments).

$$B = B \cdot GD_{cr}$$

The search continues until the bound reaches a lower limit equal to ${GD_{lb}}^{at} \cdot S_{best}$, where GD_{lb} is a parameter defining lower bound coefficient (set to 0.95 in all the experiments). When this lower limit is reached, the bound is reset back to its upper limit of ${GD_{ub}}^{at} \cdot S_{best}$ (Lines 10-11).

$$B < {GD_{lb}}^{at} \cdot S_{best} \;\Rightarrow\; B = {GD_{ub}}^{at} \cdot S_{best}$$

The parameter at is a counter starting at 1. It is increased by one every time the lower limit is reached and the bound is increased. It is also reset back to 1 when a previous best solution is improved upon. This helps the solver to widen the search when it cannot find an improvement, allowing it to get out of a deep local minimum.

1: **function** GD(σ)
2: $S_{best} = penaly(\sigma),\ B = S_{best} \cdot GD_{ub},\ \omega = \sigma,\ at = 1$
3: **while not** TIMEOUT **do**
4: $n =$ RANDOM({PERIODCHANGE, ROOMCHANGE, EXAMSWAP, RANDOMMOVE})
5: $\delta =$ GENERATE(σ, n), $S = penalty(\sigma \otimes \delta)$
6: **if** $S \leq B$ **then**
7: $\sigma = \sigma \otimes \delta$
8: **if** $S < S_{best}$ **then** $S_{best} = S,\ \omega = \sigma,\ at = 1$
9: $B = B \cdot GD_{cr}$
10: **if** $B < {GD_{lb}}^{at} \cdot S_{best}$ **then**
11: $B = {GD_{ub}}^{at} \cdot S_{best},\quad at = at + 1$
12: **return** ω

Fig. 4 Pseudo-code of the Great Deluge algorithm.

3.4 Neighborhoods

The following neighborhoods are selected with equal probability during the hill climbing and great deluge phases. All of the proposed neighborhoods attempt to change an assignment of an exam or to swap the periods and/or rooms of two exams. If a change cannot be made, it systematically searches for an alternate change by selecting the next feasible assignment that follows the initially proposed change (i.e., it tries to use one of the subsequent periods or rooms in the variable's domain in the order they are loaded from the input file) rather than randomly generating and checking some other change.

3.4.1 Period Change

An examination and a new period are randomly selected. If no conflict results from assigning the selected exam to the new period, the new assignment is returned. The following periods are tried otherwise. The first available period is returned, another neighborhood is tried if no such period can be found. For each period, it tries to keep the existing room assignment, but if the room or rooms are not available, it tries to find the best possible available room or rooms instead. If no such room assignment can be found, the search continues with the following period.

The best possible room assignment is found by looking at all the rooms that are available to the exam at a certain examination period, with the smallest number of splits that does not violate any hard constraints (e.g., same room distribution constraint) and that minimizes the weighted sum of room penalty, number of violated soft same / different room distribution constraints and the room split distance.

3.4.2 Room Change

An examination and a new room are randomly selected. A new placement is generated by using the current period, the randomly selected room. If the selected room is not big enough, some of the current rooms (but not all) are added into the room placement. If no conflict results from assigning the selected exam into the new room assignment while keeping its period assignment, the new assignment is returned. The other combinations of rooms (if there are multiple using the randomly selected room and some of the current ones) are tried otherwise. If there is no other combination, the search continues with the following room.

3.4.3 Examination Swap

Two examinations are randomly selected. A new placement is generated by swapping periods of the two exams. For each exam, the best possible room placement is found. If the two exams are in the same period, it just tries to change the room assignments by looking for the best available room placement ignoring the existing room assignments of the two exams. If no conflict results from the swap the assignment is returned. The following exams of the second exam in the pair are tried for an exam swap otherwise.

3.4.4 Random Move

An examination and a new period are randomly selected. The best possible room assignment is found for the new period. If there is such a room assignment found and no conflict results from assigning the selected exam to the new period and room(s), the new assignment is returned. The following periods are tried otherwise.

4 Experiments

The following experiments were computed on an Apple MacBook Pro with a 3.06 GHz Intel Core 2 Duo processor and 8 GB RAM, running Mac OS X 10.7.3 and Java 1.6.0. In all the experiments the solver was given a time limit of two hours, and the average values of ten independent runs are shown.

4.1 Data Instances

In this work, nine data sets were made available for benchmarking. They are all final examinations problems from Purdue University, for each semester starting Fall 2008 when UniTime was used for examination timetabling at the university for the first time. All the datasets can be downloaded at `http://www.unitime.org/exam_datasets.php`. In this paper, the last four data sets are discussed. The results for the remaining data set are available on the web site.

Problem	Fall 2012	Spring 2012	Fall 2011	Spring 2011
Exams	1,864	1,798	1,914	1,866
Students	33,279	31,593	33,856	31,688
Enrollments	117,271	111,355	122,386	113,224
Distribution constraints	20	13	6	1
Exams fixed in time	57	63	58	99
Exams fixed in room	24	6	70	170
Large exams (600+)	22	20	18	17
Exams needing room split	10	9	20	13
Exams with original room	1,533	1,485	1,524	1,485
Density	3.28%	3.59%	3.46%	3.52%
Available periods	28.2 ± 0.4	28.0 ± 0.5	28.2 ± 0.4	27.5 ± 0.7
Available rooms	262.9 ± 6.1	265.8 ± 3.2	256.3 ± 9.8	234.7 ± 11.8
that are big enough	143.3 ± 38.5	143.3 ± 37.1	135.1 ± 40.1	126.6 ± 37.5

Table 1 Characteristics of the last four data sets.

Table 1 show the basic properties of the four latest datasets. The number of examinations, students, student examination enrollments and the distribution constraints for each data set are presented. All of the datasets have 29 examination periods, all 2 hours long starting at 8 am, 10:30am, 1 pm, 3:30 pm, and 7 pm each day from Monday till Saturday, except of the last period on Saturday (at 7 pm). All four Saturday periods

(8 am, 10:30am, 1 pm, and 3:30 pm) are strongly discouraged (*penalty* = 4), Friday afternoon periods (1 pm, 3:30 pm, and 7 pm) are discouraged (*penalty* = 1).

In all problems there are only a few distribution constraints and all are marked as hard (they cannot be violated). Moreover, in each problem, there are a few exams pre-assigned in time and/or room. The number of large exams, those exams for 600 or more students, is not very high either, which is good as they all should fit within the first 24 periods (excluding Saturday and the last period on Friday; the density between these exams is over 80%). There are also a few exams in each problem that need to be split into at least two rooms as there is no room big enough for the exam in its domain, either because of the number of students in the exam or due to room requirements that are put on the exam. The probability of a pair of two exams having at least one student in common is shown in the *density* row. The last three rows show the domain sizes of the problems, i.e., average number of available periods and rooms to an exam. The last column only counts the rooms that are big enough for the exam to fit in without the need to be split into multiple rooms. About 43% of all exams require examination (alternative) seating.

Fall 2012	All	$\geq$ 100 seats	$\geq$ 200 seats	$\geq$ 400 seats	$\geq$ 600 seats
Rooms	347	30 (16)	12 (8)	7 (3)	2 (2)
Exams	1,864 (819)	248 (179)	87 (69)	37 (32)	22 (21)
Density	3.3%	29.6%	60.0%	81.2%	83.6%

Table 2 Number of rooms and exams for Fall 2012 of certain size.

There are about 350 rooms in the problem. Table 2 shows the tightness of the problems for large examination rooms on the data set from Fall 2012. The numbers in brackets denote rooms with examination seating of the given size or examinations requiring examination seating with the given number of students. The density of common students between exams increases dramatically for larger sized exams.

Assigning exams to examination periods while avoiding direct conflicts between students can be seen as a graph coloring problem. Each examination can be represented as a vertex, with edges between two vertices where there is at least one student in common between the two exams. Each color then correspond to one examination period. The chromatic number of such a graph, which is the smallest number of colors needed to color all the vertices in a way that there are no two vertices with the same color connected with an edge, corresponds to the smallest number of examination periods that are needed to create an examination timetable with no direct conflicts, while still ignoring all the other hard constraints of the problem. Table 3 shows the chromatic numbers for the graphs of the four data sets. It is interesting to see how the chromatic number is decreased when edges between exams that contain only up to 1, 2 or 5 students in common are removed. With only 29 examination periods in the problem, and with other hard constraints thrown in, this makes finding a (direct) conflict free schedule close to impossible. And, as we will show in Section 4.3, even having such a conflict free timetable would be impractical as the number of other student conflicts (primarily the more than two exams on a day conflicts) grows too high.

Problem	All Edges	>1 student	>2 students	>5 students
Fall 2012	27	21	17	12
Spring 2012	27	19	17	12
Fall 2011	29	20	16	13
Spring 2011	29	21	17	12

Table 3 Chromatic number of a graph with edges between exams with at least one, two, three, or six students in common.

4.2 Configurations

The data sets were tested on four different configurations. Solutions were measured using a weighted sum of the violated soft constraints. The appropriate weights are shown in the Table 4.

Weight	**production**	**base**	**color**	**split**
Direct Conflict	1,000	10^6	-	10^6
More Than 2 A Day	100	10^4	10^4	10^4
Back-To-Back	10	100	100	100
Period Penalty	1	1	1	1
Room Penalty	1	1	1	1
Room Split	10	10	10	10
Room Split Distance	0.01	0.01	0.01	0.01
Room Size	$(0.001x)^{1.1}$	$(0.001x)^{1.1}$	$(0.001x)^{1.1}$	$(0.001x)^{1.1}$
Rotation Penalty	0.0001	0.0001	0.0001	0.0001
Large Exams Penalty	$2.5 \cdot 10^6$	$2.5 \cdot 10^6$	$2.5 \cdot 10^6$	$2.5 \cdot 10^6$
Room Distance	0.0001	0.0001	0.0001	0.0001
Hard Constraint Violation	-	-	1,000	-
Exam Period Split	-	-	-	5,000

Table 4 Four weights configurations used in the experiments.

The **production** configuration contains the weights as they are used in practice by the University. The **base** configuration gives more weight to direct student conflicts to give a better comparison with the next two configurations. In the **color** configuration, the graph coloring solution is used to construct an initial assignment of exams to periods and rooms and it is not allowed to create a direct conflict. To be able to do this, however, it was necessary to add the ability to break other hard constraints, namely the ability to use a prohibited period or room by an exam or to break a distribution constraint. It was still not allowed to have multiple examinations in the same room at the same period, to exceed the room capacity, or to assign an exam into more than the given maximal number of rooms.

The last configuration (named **split**) attempts to minimize the number of direct conflicts by adding the ability to split an exam into two. In these runs, an additional neighborhood was used (along with all the other neighborhoods mentioned above) that allowed splitting a randomly selected examination into two, assigning each in a different period, and moving students freely between these two exams if the improvement in the weighted number of student conflicts (including direct, more than two on a day, and

back-to-backs) was higher than the penalization for splitting an exam **Exam Period Split**. If a randomly selected exam was already split, the neighborhood also tried to reshuffle the students between the two exams or merge the exams back together if the split was no longer needed (i.e., the weighted sum of student conflicts avoided by the split was smaller than the weight of the split).

4.3 Results

Results for all the runs are shown in the following tables. Table 5 shows the results for the four data sets using **production** configuration. This is the configuration that

Config: **production**	Fall 2012	Spring 2012	Fall 2011	Spring 2011
Direct Conflicts	79.7 ± 3.4	91.6 ± 5.8	107.2 ± 6.0	119.5 ± 6.5
More Than 2 A Day	345.2 ± 10.0	398.9 ± 11.9	448.2 ± 23.2	469.5 ± 7.9
Back-To-Back	4107.2 ± 74.5	4233.4 ± 51.5	4700.9 ± 94.5	4360.7 ± 65.9
Period Preference [%]	91.5 ± 0.3	91.5 ± 0.2	91.3 ± 0.3	91.0 ± 0.3
Room Preference [%]	74.3 ± 0.5	75.4 ± 0.5	70.1 ± 0.5	70.1 ± 0.4
Room Distance [m]	36.4 ± 2.1	35.2 ± 2.4	48.3 ± 2.9	48.9 ± 1.5
Room Split	43.0 ± 2.3	36.8 ± 1.9	94.6 ± 1.5	24.5 ± 1.4
Room Split Distance [m]	95.0 ± 11.3	69.4 ± 15.2	111.9 ± 12.9	48.1 ± 17.0
Room Size	72.3 ± 1.5	66.0 ± 2.8	52.7 ± 3.0	66.2 ± 1.9
Rotation Penalty	11.95 ± 0.05	12.09 ± 0.05	12.05 ± 0.07	11.95 ± 0.04
Large Exams Penalty	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 5 Results using **production** configuration. Average results for 10 independent runs are presented.

is used in practice at Purdue University. This gives a good proportion between direct and more than two exams on a day conflicts. The table shows the total number of direct student conflicts, more than two on a day conflicts and back-to-back conflicts. A percentage of the overall satisfaction of all the period and room penalties are displayed (100% would mean that all the examinations are assigned in a period or a room with the smallest penalty). The **period preferences** mostly correspond to discouraging Friday afternoon periods and strongly discouraging Saturday periods. The **room preferences** mostly correspond to assigning exams to their original room (a room where the class took place for all the class exams) or, if the room is not available, to the same building. The average distance to the original room if the original room was not assigned is shown in the **Room Distance** row. The number of exams being split into multiple rooms is very close to the absolute number of splits that are needed to fit all the big exams into available rooms. All the large exams were assigned before the last 5 periods.

The **Rotation Penalty** shows the average square root of the multiplication of the average period index and the assigned period index (both starting from 1). This number is minimized and it would be around 13.0 on average for an exam randomly assigned to a period with no regards to the rotation. It is a little less than a half of the number of examination periods (14.5 with 29 periods) as there are less exams timetabled during the discouraged Friday afternoon and strongly discouraged Saturday periods than on other (non discouraged) examination periods.

The effect of increasing direct conflict weight can be nicely seen in Table 6. The increase in the number of more than two on a day and back-to-back conflicts is most important. Impairment on other criteria is less profound.

Config: **base**	Fall 2012	Spring 2012	Fall 2011	Spring 2011
Direct Conflicts	32.7 ± 3.9	50.2 ± 2.9	53.9 ± 4.8	73.1 ± 4.4
More Than 2 A Day	344.8 ± 26.6	383.6 ± 18.7	545.2 ± 26.8	559.0 ± 20.3
Back-To-Back	4792.1 ± 151.2	4945.7 ± 147.5	5823.3 ± 86.6	5431.3 ± 157.9
Period Preference [%]	88.2 ± 0.4	87.5 ± 0.2	87.2 ± 0.4	87.7 ± 0.5
Room Preference [%]	72.4 ± 0.3	73.8 ± 0.5	69.0 ± 0.6	69.3 ± 0.7
Room Distance [m]	40.7 ± 1.6	35.5 ± 1.5	48.9 ± 3.4	51.1 ± 1.8
Room Split	48.5 ± 8.9	35.4 ± 4.2	58.5 ± 3.8	29.1 ± 2.8
Room Split Distance [m]	82.7 ± 16.0	73.9 ± 15.7	67.8 ± 11.0	54.0 ± 7.8
Room Size	70.0 ± 3.8	68.0 ± 1.6	51.6 ± 2.9	68.8 ± 2.2
Rotation Penalty	12.22 ± 0.04	12.42 ± 0.05	12.28 ± 0.04	12.30 ± 0.06
Large Exams Penalty	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 6 Results using **base** configuration. Average results for 10 independent runs are presented.

The configuration **color** with results in Table 7 was created in an attempt to bring the number of direct student conflicts down even more. While it was not allowed to

Config: **color**	Fall 2012	Spring 2012	Fall 2011	Spring 2011
Direct Conflicts	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
More Than 2 A Day	650.7 ± 38.0	663.3 ± 19.6	1152.6 ± 19.0	717.5 ± 20.6
Back-To-Back	6342.0 ± 133.5	6282.0 ± 94.5	6958.0 ± 145.1	6042.7 ± 82.0
Unavailable Period	12.7 ± 1.3	18.3 ± 0.9	17.4 ± 1.2	28.9 ± 1.3
Unavailable Room	10.8 ± 0.9	10.2 ± 1.0	22.6 ± 1.0	18.1 ± 1.8
Violated Distribution	2.8 ± 0.8	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Period Preference [%]	85.8 ± 0.3	86.3 ± 0.4	86.5 ± 0.3	86.7 ± 0.4
Room Preference [%]	72.5 ± 0.4	73.5 ± 0.4	69.3 ± 0.9	69.5 ± 0.6
Room Distance [m]	38.6 ± 2.6	38.5 ± 1.5	54.3 ± 3.8	55.2 ± 1.9
Room Split	19.8 ± 9.7	14.4 ± 5.4	30.7 ± 6.0	13.8 ± 5.1
Room Split Distance [m]	150.5 ± 65.0	137.1 ± 50.8	159.4 ± 46.6	167.0 ± 39.9
Room Size	77.5 ± 2.3	68.8 ± 2.9	54.6 ± 2.2	68.9 ± 3.9
Rotation Penalty	12.33 ± 0.05	12.64 ± 0.05	12.56 ± 0.04	12.40 ± 0.04
Large Exams Penalty	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	2.0 ± 0.0

Table 7 Results using **color** configuration. Average results for 10 independent runs are presented.

create a direct conflict, it was allowed to assign an examination to a period or room that was not available for the exam or to break a hard distribution constraint. Unlike in all the other runs, the solver started from a graph coloring solution and each color was randomly assigned to a period. Best available room or rooms were assigned to each exam in the same order as the exams are taken during the construction phase of other runs. After the construction phase the search resumed as with all the other runs (i.e.,

with the hill climbing phase), but it was not allowed to create any direct conflict during the run. This means that only neighborhoods which did not create a direct student conflict were allowed.

While there were no direct conflicts in any run using the **color** configuration, the number of more than two exams on a day and back-to-back conflicts went up quite substantially. The number of more than two exams on a day almost doubled compared to the **base** runs and there were at least one thousand more back-to-backs in most of the runs. Note that the number of hard constraint violations somewhat corresponds to the number of direct conflicts we had in the runs using the **base** configuration. All things considered, it is very interesting to see the effect of requiring a (direct) conflict free examination schedule on the other criteria; however, the results are not usable in practice, especially because of the broken hard constraints and the number of more than two exams on a day. At Purdue, more than two exams on a day conflicts are almost as bad as direct conflicts as in both cases a student is allowed to request an individual examination, though some students do not request an individual examination because of the number of exams on a day. Unfortunately, since a student with a conflict goes directly to his / her professor, we have no statistics on how many students actually do request an individual exam, but the numbers we are getting with the **production** configuration seem to work quite well. Certainly, there are more complaints about having Saturday exams than on student examination conflicts.

A different approach to further decreasing the number of student conflicts was tried using the **split** configuration. Here the solver is allowed not only to split an exam between multiple rooms, but also to split it between multiple examination periods. It can be seen that the number of exams that were split roughly corresponds to the number of direct conflicts somewhere in between the **base** and **production** configurations. This is because most of the direct conflicts in the solutions are between exams that have one or two students in common. On the other hand, allowing the solver to move students with any conflict (including back-to-backs) in between the exams helps to decrease the number of more than two on a day and back-to-back conflicts quite substantially. Also, the solutions of the **split** configuration are very similar to the **base** configuration solutions in all the other criteria, which are also shown in the Table 8.

Config: **split**	Fall 2012	Spring 2012	Fall 2011	Spring 2011
Direct Conflicts	0.0 ± 0.0	7.0 ± 0.0	0.0 ± 0.0	1.0 ± 0.0
More Than 2 A Day	71.3 ± 11.6	90.9 ± 8.5	96.9 ± 39.0	87.9 ± 11.9
Back-To-Back	1802.7 ± 112.0	1934.9 ± 138.1	2001.2 ± 464.3	1721.5 ± 132.9
Period Splits	64.10 ± 3.54	75.60 ± 6.35	88.80 ± 9.13	83.30 ± 5.83
Period Preference [%]	88.6 ± 0.4	88.0 ± 0.3	87.8 ± 1.5	87.8 ± 0.3
Room Preference [%]	72.3 ± 0.7	73.8 ± 0.5	64.7 ± 11.3	67.6 ± 2.5
Room Distance [m]	40.4 ± 2.5	36.3 ± 2.0	68.4 ± 55.7	56.3 ± 11.1
Room Split	46.8 ± 3.6	37.1 ± 6.5	55.0 ± 12.8	31.2 ± 6.7
Room Split Distance [m]	41.0 ± 10.1	37.2 ± 12.6	63.2 ± 34.5	57.5 ± 19.2
Room Size	84.1 ± 6.3	74.3 ± 2.7	70.7 ± 14.9	79.6 ± 3.3
Rotation Penalty	12.19 ± 0.04	12.32 ± 0.05	12.34 ± 0.28	12.41 ± 0.08
Large Exams Penalty	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0

Table 8 Results using **split** configuration. Average results for 10 independent runs are presented.

The ability to split an examination in periods is interesting and gives very promising results. Besides the fact that the number of makeup exams (individual exams scheduled for students with a direct or more than two on a day conflict) seems to correspond with the automatically generated period splits, it can also be used to decrease the number of exams that are now offered on Saturday, while keeping the number of student conflicts on a manageable level. On the other hand, the need to offer certain exams during two distinct examination periods has some drawbacks as well. For instance, students on the second period can have some advantage over students taking the exam first.

Figure 5 shows how the quality of the computed solution improves during the search. The average results of 10 independent runs of the Fall 2012 problem using the **production** configuration are presented. The top most chart shows the **Overall Solution Value**, which is the sum of all the violated soft constraints, each weighted by the appropriate weight from the **production** configuration. This weighted sum is minimized during the search. Each of the following charts shows one of the objectives. The **Large Exams Penalty** is omitted from the list as this penalty was always zero across all the runs due to its high weight.

For the Fall 2012 problem and the **production** configuration, the construction phase did not take more than 10 seconds and the hill climbing phase was finished within a minute. We can see the hill climbing phase before the short plateau at the beginning of each chart. The hill climbing phase is particularly useful for setting the initial bound of the following great deluge phase and starting the phase from a good solution. Despite of that, it still takes about two minutes for the great deluge phase to find a better solution. But the improvements are quite dramatic, especially in the number of direct conflicts and more than two exams a day, which have the highest weights. The other criteria get quite disturbed by these improvements at first (as in many cases, decreasing the number of student conflicts goes against all the other criteria). However, as the bound goes down, the other criteria get improved as well and in about 10 minutes (when the bound reaches the lower limit for the first time) the solution looks quite good in all of its characteristics. In fact, the **Overall Solution Value** of the solution computed after the first 15 minutes is on average only about 6.5% higher than the overall value we get after the full two hours. On the other hand, we can see small but steady improvements in the solution quality all the way during the remaining search. The lower limit of the bound got reached 18 times on average during the search and the counter *at* that widens the limits for the bound (see Figure 4) did not get higher than 3.

The experiment also validates the current practice at Purdue University. While 15 to 30 minutes works well for all the trial runs that are done during the data entry (e.g, to validate that the entered requirements do not over-constraint the problem or create too many student conflicts), the two hour limit gives a good base for a production timetable. Once a production timetable is created and published, there is rarely a need for a new run. On the other hand, UniTime allows for manual modifications of the timetable in an interactive mode, where the solver does not make any decisions but it is used to provide alternatives and suggestions how to fix created problems. For instance, if an operator wants to move an exam into a different room, it will provide suggestions how to move the other affected exams around so that there are no hard constraints violated and the quality of the modified solution (especially the number of student conflicts) is maintained.

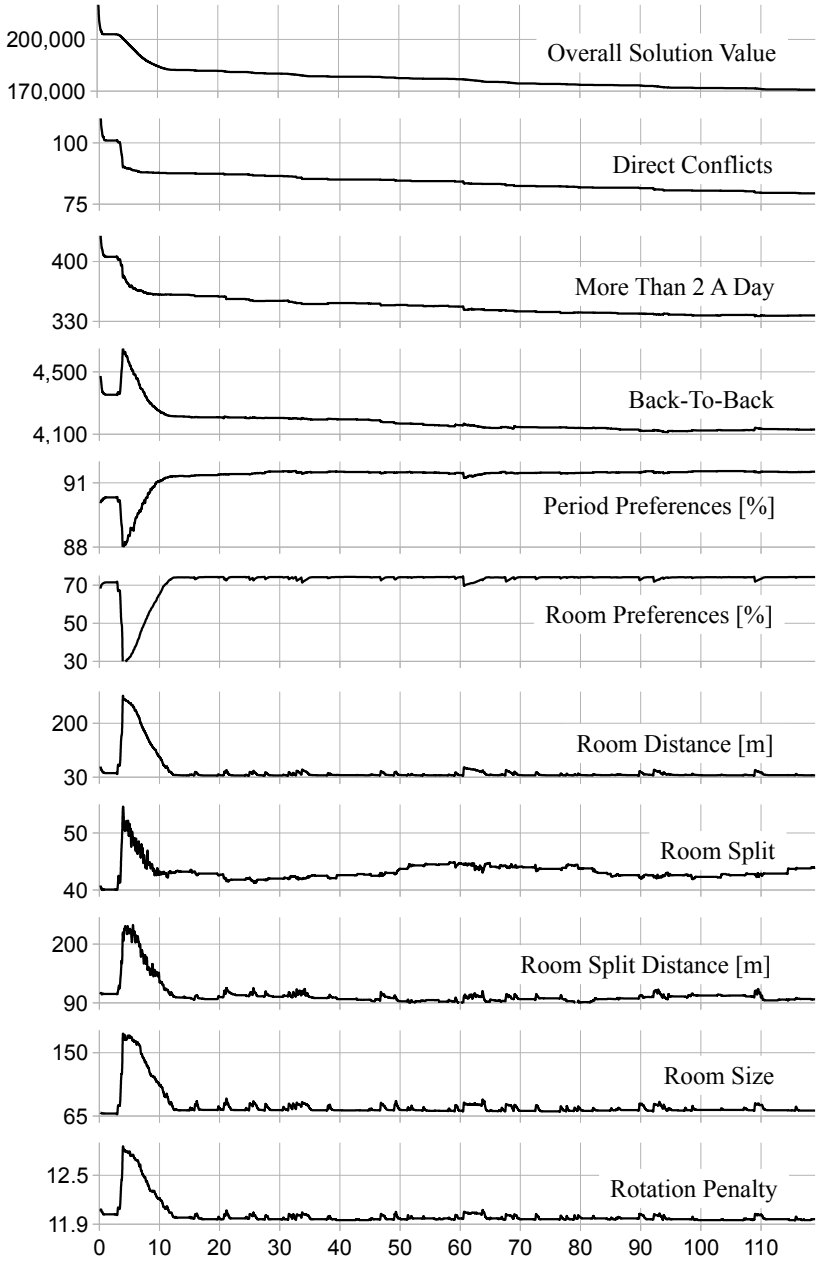


Fig. 5 Progression of the best computed solution during the search. Each chart represents one objective, the horizontal axe represents time in minutes. Average results of 10 independent runs on the Fall 2012 problem using **production** configuration are presented.

5 Conclusion

In this paper, we have presented a real world examination timetabling problem and an algorithm that was implemented in the university timetabling system UniTime.

Though we only have data from Purdue University at the moment, we believe it is applicable to many other American universities (especially the larger ones that need to deal with large exams offered to students across multiple curricula) and we plan to extend the benchmark suite that was created in this work with data from other institutions in the future. There are several other universities that are using UniTime for examination timetabling or that are in the process of adopting UniTime.

From the research perspective, the ability to split an exam into multiple examination periods seems to be very interesting. As for the future work, several other features have also been added recently to the examination solver; however, we do not have enough data to publish results at the moment. For instance, besides being able to split an exam into multiple rooms, it is now also possible to put multiple examinations into one room during the same examination period. This room sharing ability is controlled by a compatibility metrics that can be a part of the input data. For instance, only certain rooms can be shared between multiple examinations, only exams that are of the same length can share a room, and the room has to be big enough to hold all the exams that are placed in there at the same time.

References

Carter MW, Laporte G, Lee SY (1996) Examination timetabling: Algorithmic strategies and applications. Journal of the Operational Research Society 47(3):373–383, DOI 10.1057/jors.1996.37

Dueck G (1993) New optimization heuristics: The great deluge algorithm and the record-to record travel. Journal of Computational Physics 104:86–92

McCollum B, McMullan P, Parkes AJ, Burke EK, Qu R (2012) A new model for automated examination timetabling. Annals of Operations Research 194:291–315, DOI 10.1007/s10479-011-0997-x

Müller T (2005) Constraint-based timetabling. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, URL `http://muller.unitime.org/phd-thesis.pdf`

Müller T (2009) Itc2007 solver description: a hybrid approach. Annals of Operations Research 172:429–446, DOI 10.1007/s10479-009-0644-y

Müller T, Murray K (2010) Comprehensive approach to student sectioning. Annals of Operations Research 181:249–269

Müller T, Rudová H (2012) Real-life curriculum-based timetabling. In: Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling – PATAT 2012, pp 57–72

Müller T, Barták R, Rudová H (2004) Conflict-based statistics. In: Gottlieb J, Silva DL, Musliu N, Soubeiga E (eds) EU/ME Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics, University of Nottingham

Qu R, Burke E, McCollum B, Merlot L, Lee S (2009) A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12:55–89, DOI 10.1007/s10951-008-0077-5

Rudová H, Müller T, Murray K (2011) Complex university course timetabling. Journal of Scheduling 14(2):187–207

MISTA 2013

A polynomial time OPT-1 algorithm for a scheduling problem with two delivery dates and cumulative payoffs

Yasmina Seddik · Christophe Gonzales · Safia Kedad-Sidhoum

Abstract We consider a single machine scheduling problem with two common delivery dates and unequal release dates for the jobs. The goal is to maximize the cumulative number of jobs processed before each delivery date. The general problem (with $K \geq 2$ delivery dates) results from a practical situation in book digitization. The two delivery dates case considered here is weakly NP-hard. In this paper, we show for this problem a polynomial time algorithm with an absolute guarantee of 1. By considering the two delivery dates problem, this paper lays the basis for an approximation algorithm for the general K delivery dates problem.

1 Introduction

The problem considered in this paper comes from a real world problem in bibliographical digitization, where a huge amount of books must be digitized, following a linear process. We consider the one-machine problem, as a simplified way to model the linear process, or as a possible bottleneck machine. Two entities are involved: the client (the French National Library) and the manufacturer. We focus on the manufacturer's point of view, which tries to meet the client requirements while maximizing his own profits. The books to be digitized become available at different dates, and the client sets some delivery dates with related targets (number of books to be digitized). The payoff that a given job yields to the manufacturer depends on the date at which this job is delivered: the earlier the date, the greater the payoff. More details on the application can be found in previous works [11, 12].

We model the problem as follows. N jobs $\{J_1, \ldots, J_N\} = \mathcal{J}^{all}$ have to be nonpreemptively scheduled on a single machine. Each job J_i has a release date $r_i \geq 0$ and a

Yasmina Seddik
Laboratoire d'Informatique de Paris 6
E-mail: Yasmina.Seddik@lip6.fr

Christophe Gonzales
Laboratoire d'Informatique de Paris 6
E-mail: Christophe.Gonzales@lip6.fr

Safia Kedad-Sidhoum
Laboratoire d'Informatique de Paris 6
E-mail: Safia.Kedad-Sidhoum@lip6.fr

processing time $p_i > 0$. K delivery dates are given: $D_1, \ldots, D_K$, with $0 < D_1 < \cdots < D_K$. All data are integer.

Given a schedule S, which is characterized by the completion times of all the jobs, variable V_k represents the number of jobs completing at or before D_k in S. The payoff of a given schedule S is defined by $v(S) = \sum_{k=1}^{K} V_k$. The payoffs related to the delivery dates are cumulative: jobs delivered at a given delivery date are also counted for the payoffs related to each of the subsequent delivery dates. Extending the three-field notation of Graham et al. [4], this problem is denoted as $1|r_i|\sum_{k=1}^{K} V_k$.

We refer to the payoff related to a job J_i as $v(J_i)$. For a given schedule S, $v(J_i)$ is the number of delivery dates at or before which J_i completes in S. Let $C_i(S)$ (C_i when no ambiguity is possible) be the completion time of job J_i in S; $v(J_i)$ can be represented by the following non-increasing stepwise function:

$$v(J_i) = \begin{cases} K \text{ if } 0 < C_i \leq D_1 \\ \vdots \\ 2 \text{ if } D_{K-2} < C_i \leq D_{K-1} \\ 1 \text{ if } D_{K-1} < C_i \leq D_K \\ 0 \text{ if } D_K < C_i \end{cases}$$

The problem $1|r_i|\sum_{k=1}^{K} V_k$, which is strongly NP-hard [11], considers a new criterion, the cumulative payoffs. Finding an approximation algorithm for this general problem is out of the scope of this paper. However, the algorithm given in this paper lays the foundations for its development.

In this paper, we consider the special case where there are $K = 2$ delivery dates. Considering only two delivery dates remains consistent with the application: the difficulty of the problem comes rather from the great amount of books to be digitized, and two delivery dates remains a reasonable horizon to consider. The problem $1|r_i|V_1 + V_2$ is weakly NP-hard [11]. For this problem, we propose here a polynomial algorithm that yields a solution at least equal to $OPT - 1$, and whose complexity is $O(NlogN)$. To the best of our knowledge, this is the first approximation algorithm for a scheduling problem with stepwise payoff/cost functions.

A few works consider scheduling problems with stepwise payoff/cost functions.

Most of them do not take into account release dates, however the considered stepwise cost functions are job related. Detienne et al. [3] consider the single machine problem without release dates and with job related stepwise cost functions, for which they provide exact methods. Curry and Peters [1] deal with an online problem on parallel machines with reassignments. Janiak and Krysiak [7] and Tseng et al. [13] consider the single machine problem without release dates and with job related stepwise payoff functions, which they solve with heuristic methods.

Some works consider the special case where the breakpoints of the stepwise functions are common to all jobs. The single machine problem where the breakpoints are common to all the jobs is considered by Yang [14], which provides a Branch and Bound method. For the same problem, where additionally the number of breakpoints is fixed, Janiak and Krysiak [7] provide a pseudopolynomial time algorithm. Janiak and Krysiak [8] propose several heuristics to solve the problem with unrelated parallel processors.

The works cited above do not consider release dates. Notice that the problem addressed in this paper becomes polynomial if release dates are not considered [11]. As

for problems considering job related stepwise cost functions and release dates, Detienne et al. [2] propose a column generation approach for solving the parallel machines problem, while Sahin and Ahuja [10] propose mathematical programming and heuristic approaches for the single machine problem. Seddik et al. [11] consider a single machine problem with release dates and a common stepwise payoff function for the jobs: complexity results are established, and a pseudopolynomial time algorithm for the problem with two delivery dates is proposed.

The main contribution of this paper is to propose a new approximation algorithm for the two delivery dates problem.

The rest of this paper is organized as follows. In Section 2 are given some general definitions and notations. In Section 3 is described SDD-algorithm, on which is based the approximation algorithm. Section 4 presents the polynomial time approximation algorithm for $1|r_i|V_1+V_2$ and the proof of its absolute approximation guarantee. Finally, we draw some conclusions in Section 5.

2 Definitions and notations

In this section, we give some general definitions and notations.

Definitions.

- A *schedule* is a set of completion times, one for each job of $\mathcal{J}^{all}$.
- A *partial schedule* is a set of completion times, one for each job of a given set $\mathcal{J} \subset \mathcal{J}^{all}$.
- A *subschedule* of a given schedule S is a partial schedule S' where the completion times of the jobs of S' are the same in both S and S'.
- A *block* is a (partial) schedule where there are no idle times, i.e. each job of the block (except the first one) starts at the completion time of the previous job.
- In a schedule S, a *straddling job* J_i is a job such that $C_i(S) - p_i < D_k < C_i(S)$, for some $k \in \{1, 2\}$

Notations.

- We set $D_0 = 0$ and $D_3 = \max\{D_2, \max_{i=1,\dots,N} r_i\} + \sum_{i=1}^{N} p_i$.
- For any (partial) schedule S (resp. sequence of jobs Γ), we denote by $\mathcal{J}(S)$ (resp. $\mathcal{J}(\Gamma)$) the set of jobs of S (resp. Γ).
- For any (partial) schedule S, we denote by $C_{\max}(S)$ (resp. $B(S)$) the completion time of the last job of S (resp. the starting time of the first job of S).
- For any (partial) schedule S (resp. sequence of jobs Γ, resp. set of jobs $\mathcal{J}$), we denote by $p(S)$ (resp. $p(\Gamma)$, resp. $p(\mathcal{J})$) the sum of the processing times of the jobs of S (resp. Γ, resp. $\mathcal{J}$).
- Any (partial) schedule S of $1|r_i|V_1+V_2$ can be split into three subschedules S_1, S_2, S_3; S_k being the subschedule of the jobs completing into $I_k =]D_{k-1}, D_k]$, $k = 1, 2, 3$. Thus, S can be expressed as $S = S_1.S_2.S_3$, where $S_i.S_j$ denotes the concatenation of subschedule S_i with subschedule S_j (assuming that $\mathcal{J}(S_i) \cap \mathcal{J}(S_j) = \emptyset$). Moreover, if S_k is empty, we set $C_{\max}(S_k) = B(S_k) = D_{k-1}$, $k = 1, 2, 3$.

Definition. The following total order, called ERD order (for Earliest Release Date and shortest processing time) is defined on the jobs of $\mathcal{J}^{all}$. Given two jobs J_i and J_j, we denote by $J_i \prec_{ERD} J_j$ the assertion that J_i precedes J_j in the ERD order.

- $(r_i < r_j) \Rightarrow J_i \prec_{ERD} J_j$;
- $(r_i = r_j$ and $p_i < p_j) \Rightarrow J_i \prec_{ERD} J_j$;
- $(r_i = r_j$ and $p_i = p_j$ and $i < j) \Rightarrow J_i \prec_{ERD} J_j$;

The third condition is clearly arbitrary, in order to break ties and to ensure that ERD is a total order. This will be important to avoid the need of treating equivalent situations.

3 SDD-algorithm

In this section, we introduce the polynomial time Single Delivery Date algorithm (SDD-algorithm) that solves the problem SDD_I, defined below.

Problem SDD_I. Given an interval $[I_{low}, I_{up}]$ and a set of jobs $\mathcal{J}$, a feasible solution is a (partial) schedule S where each job $J_i \in \mathcal{J}(S) \subseteq \mathcal{J}$ must start at or after $\max(r_i, I_{low})$ and complete at or before I_{up}. An optimal solution is a feasible solution with the maximum number of jobs in the interval $[I_{low}, I_{up}]$.

Remark: Notice that single delivery date problem $1|r_i|V$ is equivalent to SDD_I when $I_{low} = 0$, $I_{up} = D$ (with D the unique delivery date of $1|r_i|V$) and $\mathcal{J} = \mathcal{J}^{all}$.

NOTATIONS. For any sequence Γ of jobs, and any job J_q:

- $\forall J_q \notin \mathcal{J}(\Gamma)$, $J_q.\Gamma$ denotes the sequence obtained by prepending J_q to Γ.
- $\forall J_q \in \mathcal{J}(\Gamma)$, $\Gamma \backslash J_q$ denotes the sequence obtained by removing J_q from Γ, leaving all the other jobs in the order given by Γ.
- $sched(\Gamma)$ denotes the (partial) schedule that schedules the jobs of Γ, in the order given by Γ, without idle times and with the last job completing at I_{up}.

In SDD-algorithm, depicted in Algorithm 1, the jobs of $\mathcal{J}$ (input of SDD_I) are ordered w.r.t. ERD order, i.e. $J_{i_1} \prec_{ERD} \cdots \prec_{ERD} J_{i_{|\mathcal{J}|}}$. At each iteration m ($m = |\mathcal{J}|, \ldots, 1$), the SDD-algorithm constructs a sequence Γ_{i_m} of jobs such that $sched(\Gamma_{i_m})$ is feasible (i.e. each job $J_{i_j} \in \Gamma_{i_m}$ is scheduled into $[\max(r_{i_j}, I_{low}), I_{up}]$). Moreover, in each sequence Γ_{i_m}, $m = |\mathcal{J}|, \ldots, 1$, the jobs are ordered w.r.t. ERD order. At the end of the algorithm, $sched(\Gamma_{i_1})$ is returned. An example of the algorithm execution is given in Figure 1.

It is worth noting that SDD-algorithm is close to the Moore-Hodgson algorithm [9] for solving $1||\sum U_i$.

The SDD-algorithm runs in $O(NlogN)$ if a heap is used for the search of the job with the largest processing time. Some useful properties on SDD-algorithm are given below.

Property 1 Given a set $\mathcal{J}$ of jobs and an interval $[I_{low}, I_{up}]$, the SDD-algorithm produces a partial schedule that schedules the maximal number $N(\mathcal{J}, I_{low}, I_{up})$ of jobs of $\mathcal{J}$ between I_{low} and I_{up}.

Input: $\mathcal{J} = \{J_{i_1}, \ldots, J_{i_{|\mathcal{J}|}}\}$, I_{low}, I_{up}
Output: S
$\Gamma_{i_{|\mathcal{J}|+1}} \leftarrow \emptyset$, $t \leftarrow I_{up}$
for $m = |\mathcal{J}|$ **to** 1 **do**
  $\Gamma_{i_m} \leftarrow J_{i_m}.\Gamma_{i_{m+1}}$
  $t \leftarrow t - p_{i_m}$
  if $t < \max\{r_{i_m}, I_{low}\}$ **then**
    $q \leftarrow \max\{l | p_{i_l} = \max\{p_{i_j} | J_{i_j} \in \mathcal{J}(\Gamma_{i_m})\}\}$
    $\Gamma_{i_m} \leftarrow \Gamma_{i_m} \backslash J_{i_q}$
    $t \leftarrow t + p_{i_q}$
return $S \leftarrow sched(\Gamma_{i_1})$

Algorithm 1: SDD-algorithm.

Property 2 Given a set $\mathcal{J}$ of jobs and an interval $[I_{low}, I_{up}]$, the SDD-algorithm produces a partial schedule with the shortest processing time $p(\mathcal{J}, I_{low}, I_{up})$ among all the feasible partial schedules of $N(\mathcal{J}, I_{low}, I_{up})$ jobs of $\mathcal{J}$ between I_{low} and I_{up}.

For the arguments of the proofs of Properties 1 and 2, refer to the proof of optimality of SDD-algorithm [11].

Property 3 The SDD-algorithm produces a partial schedule where the jobs are ordered w.r.t. ERD order.

Property 4 When, in the SDD-algorithm, a job is removed from the current sequence, the job with the longest processing time is chosen, or the job with the highest ranking in ERD order in case of a tie.

Properties 3 and 4 are true by construction.

4 Approximation algorithm

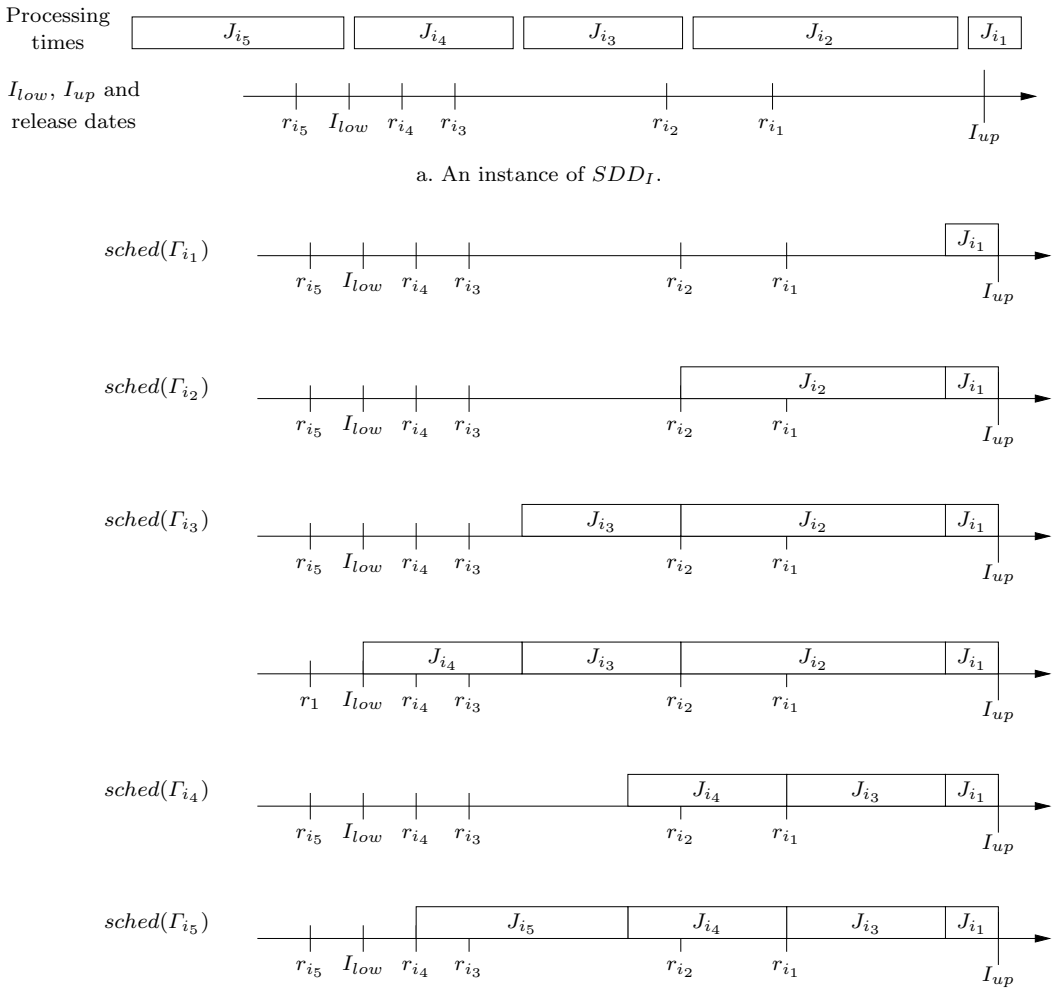
In this section, we first present the polynomial time algorithm APPROX yielding a feasible schedule, then we show how to compute an upper bound on the value of the optimal solution, and finally, we prove that the difference between the payoff of the feasible schedule obtained with APPROX and the upper bound is at most 1.

4.1 Algorithm APPROX

Algorithm APPROX, described in pseudocode Algorithm 2, yields a feasible schedule $S^l = S_1^l.S_2^l.S_3^l$, as stated below. For any feasible (partial) schedule S, we denote by left-shift(S) the schedule S after left-shifting all its jobs, i.e. where each job starts as soon as possible, either at its release date or at the completion time of the preceding job in the schedule.

Since the complexity of SDD-algorithm is $O(NlogN)$, and the left-shifting operation is performed in $O(N)$ time, the complexity of algorithm APPROX is $O(NlogN)$.

Moreover, algorithm APPROX produces a feasible schedule for $1|r_i|V_1+V_2$. Indeed, SDD-algorithm yields feasible subschedules, and left-shift operations also maintain feasibility.



a. An instance of SDD_I.

b. Iterations of SDD-algorithm on the given instance.

Fig. 1: Execution of SDD-algorithm on an instance of SDD_I.

4.2 Computing an upper bound

To prove the approximation guarantee, we will compare the payoff of the solution yielded by algorithm APPROX to the upper bound obtained with Algorithm 3.

Proposition 1 *Algorithm 3 returns an upper bound on the optimal payoff* $V_1 + V_2$ *for* $1|r_i|V_1 + V_2$.

Proof Let V_k^* be the number of jobs completing in $[0, D_k]$, $k = 1, 2$, in an optimal schedule S^*. Algorithm 3 computes U_k as the number of jobs of $SDD(\mathcal{J}^{all}, 0, D_k)$, which is the maximal number of jobs that can complete in $[0, D_k]$, $k = 1, 2$, in any

Input: $\mathcal{J}^{all}$, D_1, D_2
Output: S^l, l_bound
$\mathcal{J} \leftarrow \mathcal{J}^{all}$
$S_1^l \leftarrow SDD(\mathcal{J}, 0, D_1)$
$S_1^l \leftarrow$ left-shift(S_1^l)
$\mathcal{J} \leftarrow \mathcal{J} \backslash \mathcal{J}(S_1^l)$
$S_2^l \leftarrow SDD(\mathcal{J}, C_{max}(S_1^l), D_2)$
$\mathcal{J} \leftarrow \mathcal{J} \backslash \mathcal{J}(S_2^l)$
$S_3^l \leftarrow$ a (partial) schedule that schedules the jobs of $\mathcal{J}$ in any order after D_2
$l_bound \leftarrow 2|\mathcal{J}(S_1^l)| + |\mathcal{J}(S_2^l)|$
return $S_1^l.S_2^l.S_3^l$, l_bound

Algorithm 2: Algorithm APPROX.

Input: $\mathcal{J}^{all}$, D_1, D_2
Output: u_bound
$S^{D_1} \leftarrow SDD(\mathcal{J}^{all}, 0, D_1)$
$U_1 \leftarrow |\mathcal{J}(S^{D_1})|$
$S^{D_2} \leftarrow SDD(\mathcal{J}^{all}, 0, D_2)$
$U_2 \leftarrow |\mathcal{J}(S^{D_2})|$
$u_bound \leftarrow U_1 + U_2$
return u_bound

Algorithm 3: Computation of an upper bound on the optimal payoff of $1|r_i|V_1+V_2$.

feasible schedule, by Property 1. Therefore, $V_k^* \leq U_k$, $k = 1, 2$. Hence, $v(S^*) = V_1^* + V_2^* \leq U_1 + U_2 = u_bound$, where u_bound is the value computed by Algorithm 3. □

4.3 The approximation guarantee

In this section, we prove the approximation guarantee.

Notations.

- Let S^l be the feasible schedule obtained with algorithm APPROX. Let $S_{1,2}^l = S_1^l.S_2^l$ be the partial schedule equal to S^l between 0 and D_2.
- We denote by S^{D_k}, the (partial) schedule obtained by applying the SDD-algorithm on the jobs of $\mathcal{J}^{all}$, on the interval $[0, D_k]$: $S^{D_k} = SDD(\mathcal{J}^{all}, 0, D_k)$, $k = 1, 2$.
- We call U_k the number of jobs of S^{D_k}: $U_k = |\mathcal{J}(S^{D_k})|$, $k = 1, 2$. We denote by $u_bound = U_1 + U_2$ the upper bound yielded by Algorithm 3.
- For any job $J_i \in \mathcal{J}^{all}$, we denote by $\mathcal{J}_i^+$ (resp. $\mathcal{J}_i^-$) the set of jobs of $\mathcal{J}^{all}$ that follow J_i in ERD order, i.e with greater indices (resp. that precede J_i in ERD order, i.e with smaller indices).

Notice that $v(S_{1,2}^l) = v(S^l)$, because the jobs completing after D_2 do not provide any payoff.

For the sake of readability, from now on jobs are reindexed in ERD order: $\mathbf{J_1} \prec_{\mathbf{ERD}} \cdots \prec_{\mathbf{ERD}} \mathbf{J_N}$.

The main result of this section is stated in Theorem 1.

Theorem 1 *Algorithm APPROX yields a feasible schedule S^l for $1|r_i|V_1 + V_2$ such that $v(S^l) \geq v(S^*) - 1$, where S^* is an optimal schedule for $1|r_i|V_1 + V_2$.*

Sketch of proof. We prove that $u_bound - v(S^l) = u_bound - v(S^l_{1,2}) \leq 1$.

We have: $v(S^l_{1,2}) = 2|\mathcal{J}(S^l_1)| + |\mathcal{J}(S^l_2)| = |\mathcal{J}(S^l_1)| + |\mathcal{J}(S^l_{1,2})|$. Moreover, $\mathcal{J}(S^l_1) = \mathcal{J}(S^{D_1})$ by construction of these two schedules (cf. lines 1-2 of Algorithm 2 and line 1 of Algorithm 3). Hence, $v(S^l_{1,2}) = |\mathcal{J}(S^{D_1})| + |\mathcal{J}(S^l_{1,2})| = U_1 + |\mathcal{J}(S^l_{1,2})|$. Furthermore, $u_bound = U_1 + U_2$. Hence, $u_bound - v(S^l_{1,2}) = U_2 - |\mathcal{J}(S^l_{1,2})|$.

We show next that $U_2 - |\mathcal{J}(S^l_{1,2})| \leq 1$ (Proposition 2). □

Proposition 2 $U_2 - |\mathcal{J}(S^l_{1,2})| \leq 1$.

Before giving the formal proof of Proposition 2, let us give its rationale.

Notice that we are comparing the number of jobs of two feasible schedules: U_2 is the number of jobs of $S^{D_2} = SDD(\mathcal{J}^{all}, 0, D_2)$, and $|\mathcal{J}(S^l_{1,2})|$ is the number of jobs of $S^l_{1,2}$, constructed with algorithm APPROX.

Let us consider the example of Figure 2. On Figure 2a is given an instance of $1|r_i|V_1 + V_2$, with 5 jobs J_1, J_2, J_3, J_4, J_5. On Figure 2b is represented schedule $S^l_{1,2}$ obtained with algorithm APPROX on the given instance; while on Figure 2c is represented schedule S^{D_2} obtained as $S^{D_2} = SDD(\mathcal{J}^{all}, 0, D_2)$. The meaning of the "hypothetical" schedule of Figure 2d is explained below.

On this example, we have $U_2 - |\mathcal{J}(S^l_{1,2})| = 1$. Let us give an intuitive reason for which we cannot have $U_2 - |\mathcal{J}(S^l_{1,2})| \geq 2$. We can see on Figure 2 that the presence of an additional job in S^{D_2} is due to a straddling job. Indeed, the number of jobs completing in $]0, D_1]$ in S^{D_2} cannot be greater than $|\mathcal{J}(S^l_1)|$, by Property 1 on S^l_1. Hence, the only way S^{D_2} can schedule more jobs than $S^l_{1,2}$ is to have more jobs completing in $]D_1, D_2]$. However, if in S^{D_2} there were two more jobs completing in $]D_1, D_2]$ than in $S^l_{1,2}$, one of them would be entirely scheduled in $]D_1, D_2]$, as shown in Figure 2d; in this case, this non straddling job (job J_7 in the example) would have been scheduled in $S^l_{1,2}$.

The rest of the current section is devoted to the formal proof of Proposition 2.

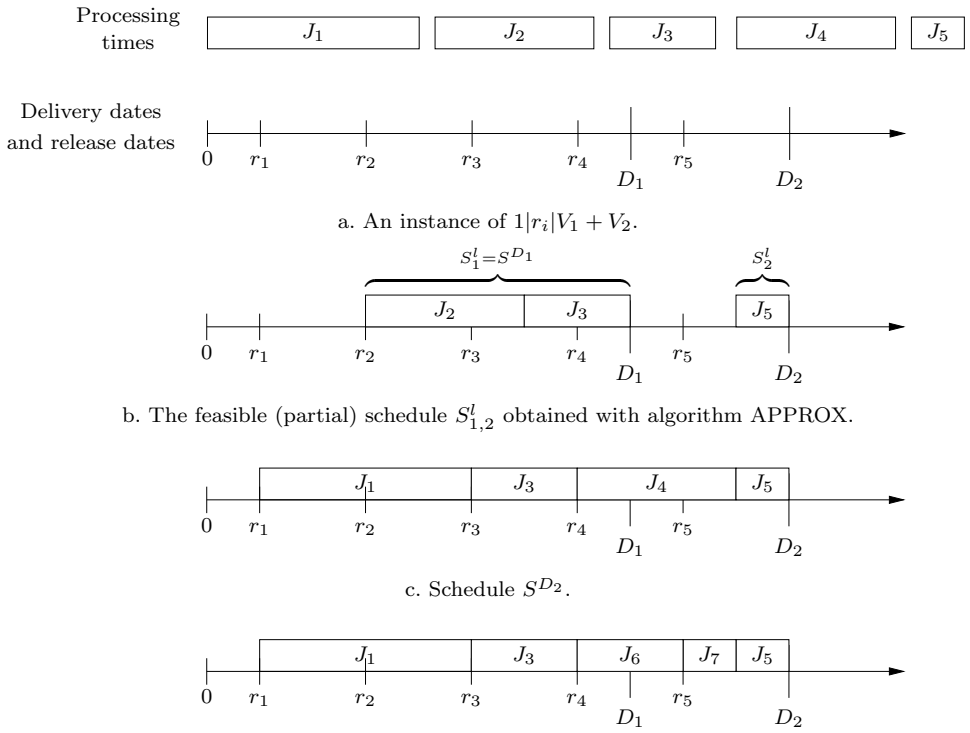
First, we prove intermediate results in Lemmas 1, 2 and 3, concerning (partial) schedules S^{D_2} and $S^l_{1,2}$. Then, we introduce another partial schedule, S^{mid}, and give its features. Finally, we prove the main result of Theorem 2.

The (partial) schedules that will be useful for the proof are the following.

- $S^{D_2} = SDD(\mathcal{J}^{all}, 0, D_2)$
- $S^l_1 = \textsf{left-shift}(SDD(\mathcal{J}^{all}, 0, D_1))$
- $S^l_2 = SDD(\mathcal{J}^{all} \backslash \mathcal{J}(S^l_1), C_{max}(S^l_1), D_2)$
- $S^l_{1,2} = S^l_1.S^l_2$

Notice that, by Property 3, the jobs are scheduled in ERD order in S^{D_2}, in S^l_1 and in S^l_2.

In Lemma 1, we show that any job J_l of S^l_2 such that all the following jobs in S^l_2 belong to S^{D_2} has a shorter or equal processing time than any job J_d of S^{D_2} that is not in $S^l_{1,2}$. The result holds, whether or not J_l belongs to S^{D_2}.



a. An instance of $1|r_i|V_1 + V_2$.

b. The feasible (partial) schedule $S^l_{1,2}$ obtained with algorithm APPROX.

c. Schedule S^{D_2}.

d. If two more jobs (J_6 and J_7) could complete in $]D_1, D_2]$, then at least one of the two jobs (J_7) would have been scheduled in $S^l_{1,2}$.

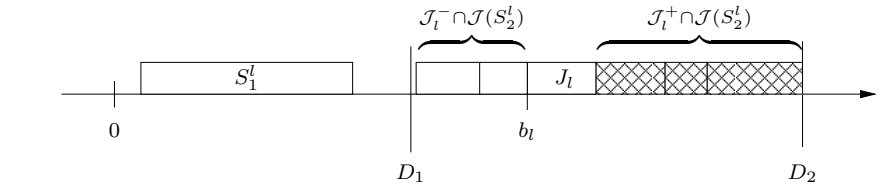
Fig. 2: An example where $U_2 - |\mathcal{J}(S^l_{1,2})| = 1$.

Lemma 1 *For any pair of jobs J_d, J_l such that:*

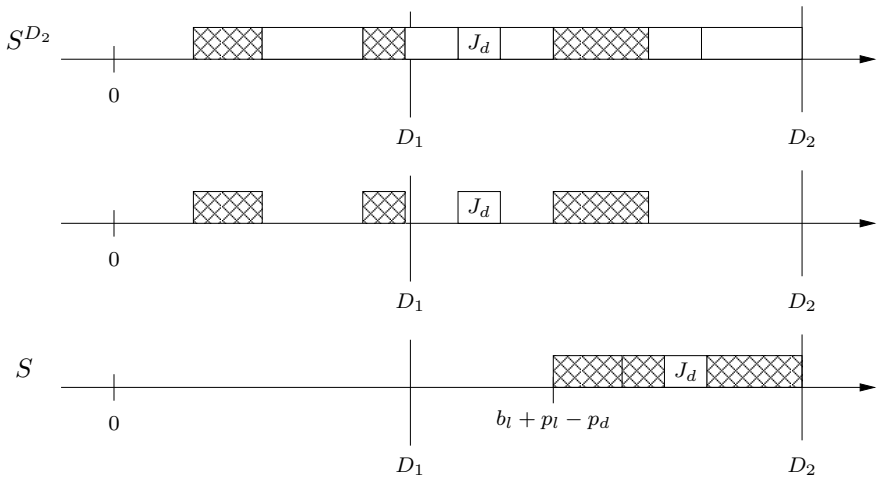
- $J_d \in \mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_{1,2})$, *and*
- $J_l \in \mathcal{J}(S^l_2)$, *and*
- $\mathcal{J}_l^+ \cap \mathcal{J}(S^l_2) \subseteq \mathcal{J}(S^{D_2})$,

we have: $p_d \geq p_l$.

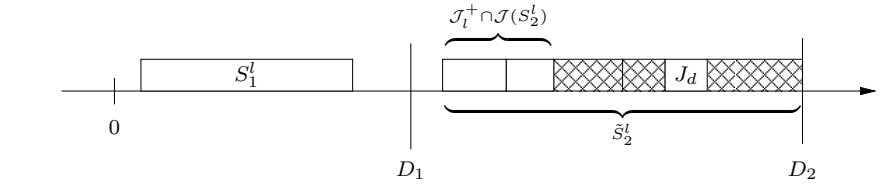
Proof By contradiction, suppose that $p_d < p_l$. By Property 3, the jobs of $\mathcal{J}_l^+ \cap \mathcal{J}(S^l_2)$ are exactly the jobs scheduled after J_l in S^l_2. Let b_l be the starting time of J_l in S^l_2: $b_l = D_2 - p(\mathcal{J}_l^+ \cap \mathcal{J}(S^l_2)) - p_l$ (see Figure 3a). Let us now consider S^{D_2}, and suppose that we remove from it all the jobs, except J_d and the jobs of $\mathcal{J}_l^+ \cap \mathcal{J}(S^l_2)$. We obtain a feasible (partial) schedule (see Figure 3b). If the jobs of the obtained schedule are then right-shifted so that the last one completes at D_2, then the schedule remains feasible. Moreover, the obtained schedule S includes J_d and the jobs of $\mathcal{J}_l^+ \cap \mathcal{J}(S^l_2)$, and starts at $b_l + p_l - p_d > b_l$, since $p_d < p_l$. Therefore, there exists a feasible partial schedule $\tilde{S}^l_2$ (see Figure 3c) without idle times and that completes at D_2, scheduling the following jobs: first the jobs of $\mathcal{J}_l^- \cap \mathcal{J}(S^l_2)$ in the same order as in S^l_2, followed by S. Moreover, $\tilde{S}^l_2$ schedules only jobs of $\mathcal{J}^{all} \backslash \mathcal{J}(S^l_1)$, between $C_{max}(S^l_1)$ and D_2; and since $\mathcal{J}(\tilde{S}^l_2) = (\mathcal{J}(S^l_2) \backslash \{J_l\}) \cup \{J_d\}$, we have that $|\mathcal{J}(\tilde{S}^l_2)| = |\mathcal{J}(S^l_2)|$, and that $p(\tilde{S}^l_2) < p(S^l_2)$, since $p_d < p_l$. This is in contradiction with Property 2 on S^l_2. □



a. (Partial) schedule $S_{1,2}^l$.



b. From top to bottom: (partial) schedule S^{D_2}, an intermediate partial schedule, partial schedule S.



c. (Partial) schedule $S_1^l.\tilde{S}_2^l$.

Fig. 3: Illustrating the proof of Lemma 1.

In Lemma 2, we show that all the jobs of S_2^l are scheduled in S^{D_2}.

Lemma 2 $\mathcal{J}(S_2^l) \subseteq \mathcal{J}(S^{D_2})$.

Proof We first show that, if there exists a job J_l in $\mathcal{J}(S_2^l) \backslash \mathcal{J}(S^{D_2})$, then $J_d \prec_{ERD} J_l$ and $p_d > p_l$, for all $J_d \in \mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S_{1,2}^l)$ (Part 1). Then, we show that $\tilde{S}^{D_2} = SDD(\mathcal{J}(S^{D_2}) \cup \{J_l\}, 0, D_2)$ is equal to S^{D_2} (Part 2). Finally, we show that J_l belongs to $\tilde{S}^{D_2}$ (Part 3). Hence, J_l belongs to S^{D_2}, which is in contradiction with the initial assumption.

Part 1

By contradiction, let us suppose that there exists at least one job in $\mathcal{J}(S_2^l)\backslash\mathcal{J}(S^{D_2})$. Then, there exists at least one job in $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S_{1,2}^l)$, since $|\mathcal{J}(S^{D_2})| \geq |\mathcal{J}(S_{1,2}^l)|$, from Property 1 on S^{D_2}.

Let J_l be the last scheduled job in S_2^l among the jobs of $\mathcal{J}(S_2^l)\backslash\mathcal{J}(S^{D_2})$. We have: $J_l \in \mathcal{J}(S_2^l)$, and all the jobs that succeed J_l in S_2^l belong to S^{D_2}: $\mathcal{J}_l^+ \cap \mathcal{J}(S_2^l) \subseteq \mathcal{J}(S^{D_2})$. Then, by Lemma 1, for any d such that $J_d \in \mathcal{J}(S^{D_2})\backslash\mathcal{J}(S_{1,2}^l)$ we have $p_d \geq p_l$.

Let us show that $d < l$ (i.e. that $J_d \prec_{ERD} J_l$). By contradiction, suppose that $d \geq l$. We cannot have $d = l$ because $J_l \in \mathcal{J}(S_{1,2}^l)$ while $J_d \notin \mathcal{J}(S_{1,2}^l)$. So, $d > l$. Then, since $d > l$ (which implies $r_d \geq r_l$) and $p_d \geq p_l$, J_d can be replaced by J_l in S^{D_2}, which yields a schedule $\widehat{S}^{D_2}$ with as many jobs as in S^{D_2} (see Figure 4). If $p_d > p_l$, then $p(\widehat{S}^{D_2}) < p(S^{D_2})$, which is in contradiction with Property 2 on S^{D_2}. Otherwise, if $p_d = p_l$, $p(\widehat{S}^{D_2}) = p(S^{D_2})$ but $\widehat{S}^{D_2}$ contains a job with a smaller index (J_l, instead of J_d in S^{D_2}), which is in contradiction with Property 4 on S^{D_2}.

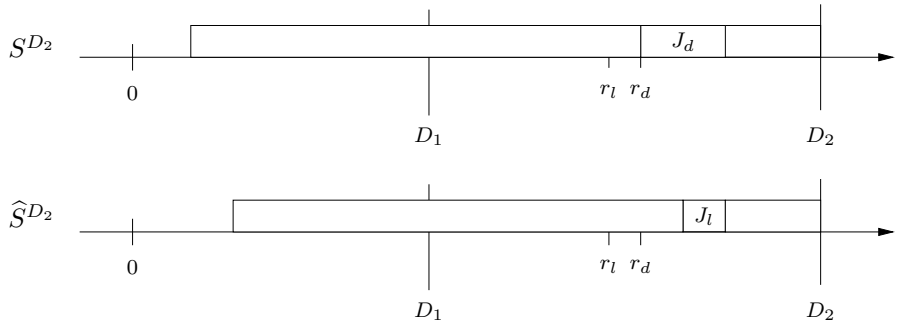


Fig. 4: Illustrating the proof of Lemma 2, Part 1: case $p_d > p_l$.

So, we have $p_d \geq p_l$ and $d < l$ (which implies $r_d \leq r_l$). Let us show that $p_d > p_l$. By contradiction, suppose that $p_d = p_l$. Then, J_l can be replaced by J_d in S_2^l (see Figure 5), which yields a schedule with as many jobs as S_2^l, whose total processing time is $p(S_2^l)$, and containing a job with a smaller index. This is in contradiction with Property 4 on S_2^l. Thus, $p_d > p_l$.

Therefore, we have, for all $J_d \in \mathcal{J}(S^{D_2})\backslash\mathcal{J}(S_{1,2}^l)$:

- $d < l$ $\quad(\alpha)$
- $p_d > p_l$ $\quad(\beta)$

Part 2

NOTATIONS FOR PART 2. b_1^+ denotes the starting time of the first scheduled job in S_1^l among the jobs of $\mathcal{J}_l^+$. b_l denotes the starting time of J_l in S_2^l.

Figure 6 shows the distribution of the jobs in $S_{1,2}^l$. From Property 3 on S_1^l and S_2^l, the jobs of S_1^l (respectively S_2^l) are scheduled in ERD order.

Let us consider the following set of jobs: $\mathcal{J}(S^{D_2}) \cup \{J_l\}$. We call $\tilde{S}^{D_2}$ the partial schedule obtained by applying the SDD-algorithm on the jobs of $\mathcal{J}(S^{D_2})\cup\{J_l\}$ between 0 and D_2: $\tilde{S}^{D_2} = SDD(\mathcal{J}(S^{D_2}) \cup \{J_l\}, 0, D_2)$. We show next that $\tilde{S}^{D_2} = S^{D_2}$.

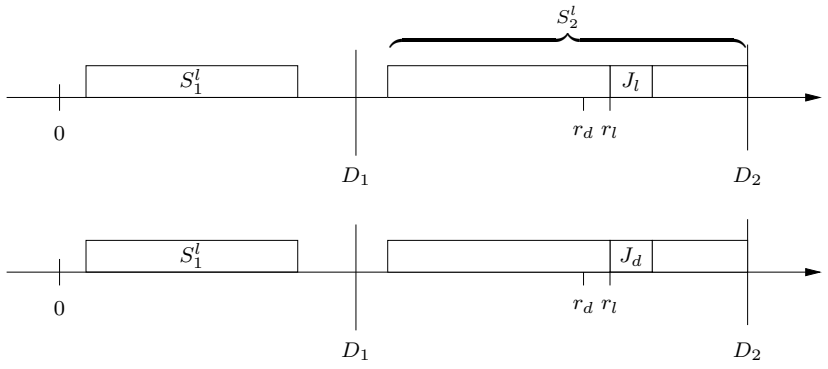


Fig. 5: Illustrating the proof of Lemma 2, Part 1.

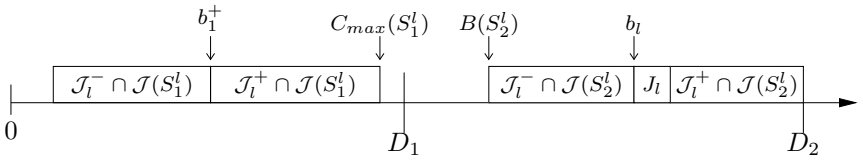


Fig. 6: The position of J_l in $S_{1,2}^l$.

Since the jobs of $\mathcal{J}(S^{D_2}) \cup \{J_l\}$ are a subset of the total set of jobs $\mathcal{J}^{all}$, no more of $|\mathcal{J}(S^{D_2})|$ of them can be scheduled between 0 and D_2, in $\tilde{S}^{D_2}$, from Property 1 on S^{D_2}. Moreover, at least $|\mathcal{J}(S^{D_2})|$ of them are scheduled in $\tilde{S}^{D_2}$ (since the jobs of $\mathcal{J}(S^{D_2})$ can be scheduled between 0 and D_2). So, $|\mathcal{J}(\tilde{S}^{D_2})| = |\mathcal{J}(S^{D_2})|$. Let us prove that $\mathcal{J}(\tilde{S}^{D_2}) = \mathcal{J}(S^{D_2})$. This will imply $\tilde{S}^{D_2} = S^{D_2}$, since both schedules complete at D_2 without idle times, and they both schedule the jobs in ERD order (i.e. increasing indices). Suppose by contradiction that $\mathcal{J}(\tilde{S}^{D_2}) \neq \mathcal{J}(S^{D_2})$. Since J_l is the only job among those of $\mathcal{J}(S^{D_2}) \cup \{J_l\}$ that is not in $\mathcal{J}(S^{D_2})$, and since $|\mathcal{J}(\tilde{S}^{D_2})| = |\mathcal{J}(S^{D_2})|$, we deduce that $J_l \in \mathcal{J}(\tilde{S}^{D_2})$ and that there exists one job in $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(\tilde{S}^{D_2})$. Moreover, from Properties 2 and 4 on $\tilde{S}^{D_2}$, one of the following assertions holds:

- $p(\mathcal{J}(\tilde{S}^{D_2})) < p(\mathcal{J}(S^{D_2}))$ (which is in contradiction with Property 2 on S^{D_2}), or
- $p(\mathcal{J}(\tilde{S}^{D_2})) = p(\mathcal{J}(S^{D_2}))$ and l is smaller than the index of the job of $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(\tilde{S}^{D_2})$ (which is in contradiction with Property 4 on S^{D_2})

Therefore, $\tilde{S}^{D_2} = S^{D_2}$.

Part 3

Now, by examining the construction of $\tilde{S}^{D_2}$, we show that $J_l \in \mathcal{J}(\tilde{S}^{D_2})$, which leads to a contradiction. Let J_{l^+} be the job of $\mathcal{J}(S^{D_2}) \cup \{J_l\}$ that immediately follows J_l in ERD order.

When constructing $\tilde{S}^{D_2}$ with the SDD-algorithm: $SDD(\mathcal{J}(S^{D_2}) \cup \{J_l\}, 0, D_2)$, the jobs are considered in reverse ERD order (i.e. decreasing indices), and are added (or not) to the current sequence, possibly by removing another job. Any job $J_d \in \mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S_{1,2}^l)$ is such that $d < l$ from (α), and is thus considered after J_l when constructing $\tilde{S}^{D_2}$. Therefore, $\mathcal{J}_l^+ \cap \mathcal{J}(S^{D_2}) \subseteq \mathcal{J}_l^+ \cap \mathcal{J}(S_{1,2}^l)$. So, when J_l is considered,

the jobs of the current sequence Γ_{l^+} (cf. SDD-algorithm, Section 3) all belong to $\mathcal{J}_l^+ \cap \mathcal{J}(S_{1,2}^l)$.

We now show that J_l can be added to Γ_{l^+} without removing any other job.

NOTATION FOR PART 3. Let B_{l^+} be the starting time of $sched(\Gamma_{l^+})$.

$B_{l^+} = D_2 - p(\mathcal{J}(\Gamma_{l^+})) \geq D_2 - p(\mathcal{J}_l^+ \cap \mathcal{J}(S_1^l)) - p(\mathcal{J}_l^+ \cap \mathcal{J}(S_2^l))$ since $\mathcal{J}(\Gamma_{l^+}) \subseteq \mathcal{J}_l^+ \cap \mathcal{J}(S_{1,2}^l)$.

Then, we need to show that J_l can be scheduled at the beginning of $sched(\Gamma_{l^+})$, i.e. $r_l \leq B_{l^+} - p_l$.

If $\mathcal{J}_l^+ \cap \mathcal{J}(S_1^l)$ is empty, then $B_{l^+} \geq D_2 - p(\mathcal{J}_l^+ \cap \mathcal{J}(S_2^l)) = b_l + p_l$ (see Figure 6). And since $b_l \geq r_l$ (otherwise $S_{1,2}^l$ would not be feasible), we have that $r_l \leq B_{l^+} - p_l$.

Conversely, if $\mathcal{J}_l^+ \cap \mathcal{J}(S_1^l)$ is not empty, then we have: $B_{l^+} \geq D_2 - p(\mathcal{J}_l^+ \cap \mathcal{J}(S_2^l)) - p(\mathcal{J}_l^+ \cap \mathcal{J}(S_1^l))$. Therefore, $B_{l^+} - p_l \geq D_2 - p(\mathcal{J}_l^+ \cap \mathcal{J}(S_2^l)) - p_l - p(\mathcal{J}_l^+ \cap \mathcal{J}(S_1^l)) = b_l - p(\mathcal{J}_l^+ \cap \mathcal{J}(S_1^l))$ (see Figure 6). Moreover, $b_l \geq B(S_2^l) \geq C_{max}(S_1^l) = b_1^+ + p(\mathcal{J}_l^+ \cap \mathcal{J}(S_1^l))$. From the inequalities above: $B_{l^+} - p_l \geq b_1^+$. Furthermore, $r_l \leq b_1^+$, because the job that starts at time b_1^+ in $S_{1,2}^l$ has a release date greater than or equal to r_l, since it belongs to $\mathcal{J}_l^+$ and $S_{1,2}^l$ is feasible. From $B_{l^+} - p_l \geq b_1^+$ and $r_l \leq b_1^+$, we deduce: $r_l \leq B_{l^+} - p_l$.

Therefore, J_l can be added to Γ_{l^+} without removing any job.

However, J_l does not belong to $\tilde{S}^{D_2}$ since $\tilde{S}^{D_2} = S^{D_2}$ and $J_l \notin \mathcal{J}(S^{D_2})$: thus, by the SDD-algorithm, there exists $J_x \in \mathcal{J}(S^{D_2})$, $x < l$, such that J_l belongs to Γ_{x^+} but not to Γ_x (i.e. J_l is removed from the current partial schedule in order to add J_x).

Suppose that J_x does not belong to $S_{1,2}^l$: $J_x \in \mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S_{1,2}^l)$. Therefore, by (β), $p_x > p_l$, which implies that J_l is not removed in order to insert J_x in S^{D_2}. So, $J_x \in \mathcal{J}(S_{1,2}^l)$.

Let us show that there exists in Γ_{x^+} at least one job that does not belong to $\mathcal{J}(S_{1,2}^l)$. By contradiction, suppose that $\mathcal{J}(\Gamma_{x^+}) \subseteq \mathcal{J}(S_{1,2}^l)$. Moreover, $J_x \in \mathcal{J}(S_{1,2}^l)$. Let us consider $S_{1,2}^l$, and remove from it all the jobs except those of $\mathcal{J}(\Gamma_{x^+}) \cup \{J_x\}$: we obtain a feasible schedule. If we reorder the jobs by increasing indices (i.e. in ERD order), and right-shift them in order to obtain a unique block that completes in D_2, we still have a feasible schedule (see Figure 7). Notice that this schedule is exactly the same obtained by adding J_x at the beginning of $sched(\Gamma_{x^+})$. Hence, no job of Γ_{x^+} is removed in order to add J_x, and in particular J_l is not removed from Γ_{x^+}. Therefore, $J_l \in \mathcal{J}(\tilde{S}^{D_2}) = \mathcal{J}(S^{D_2})$, which is in contradiction with the hypothesis $J_l \notin \mathcal{J}(S^{D_2})$. Therefore, there exists at least one job $J_d \in \mathcal{J}(\Gamma_{x^+}) \backslash \mathcal{J}(S_{1,2}^l)$.

From (β), $p_d > p_l$. Hence, there exists at least one job in Γ_{x^+} that has a longer processing time than J_l. Therefore, J_l cannot be removed in order to add J_x. Consequently, $J_l \in \mathcal{J}(\tilde{S}^{D_2}) = \mathcal{J}(S^{D_2})$, which is in contradiction with the hypothesis $J_l \notin \mathcal{J}(S^{D_2})$. □

Let us introduce the following notation, necessary for Lemma 3, and illustrated in Figure 8.

NOTATION. We set: $y = B(S_2^l) - C_{max}(S_1^l)$.

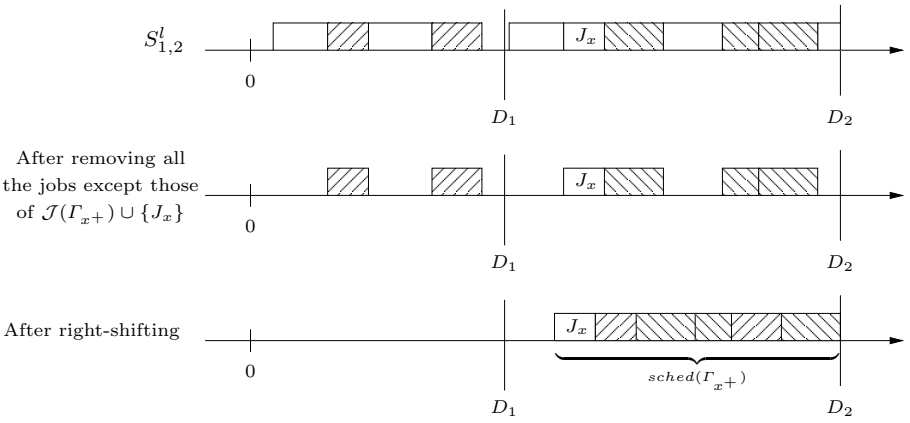


Fig. 7: Illustrating the proof of Lemma 2, Part 3: the striped jobs are the jobs of $\mathcal{J}(\Gamma_{x^+})$.

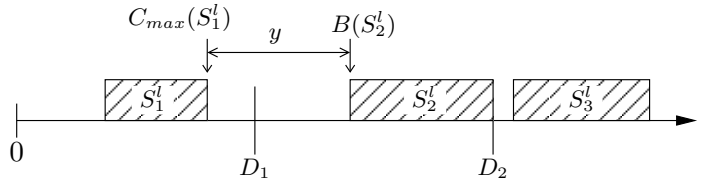


Fig. 8: Illustration of $y = B(S^l_2) - C_{max}(S^l_1)$.

Lemma 3 *For any job J_i of $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_{1,2})$, we have $p_i > y$.*

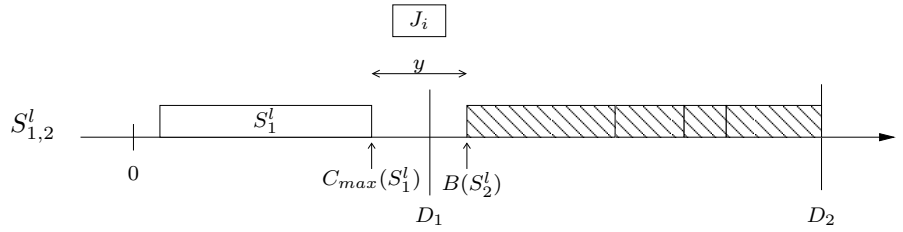
Proof By contradiction, suppose that $p_i \leq y$. Let us consider S^{D_2}, and suppose that we remove from it all the jobs, except J_i and the jobs of S^l_2 ($\mathcal{J}(S^l_2) \subseteq \mathcal{J}(S^{D_2})$ by Lemma 2). We obtain a feasible (partial) schedule. If the remaining jobs are then right-shifted so that the last one completes at D_2, the schedule still remains feasible. Moreover, the obtained schedule S includes J_i and the jobs of S^l_2, and starts at $B(S^l_2) - p_i \geq C_{max}(S^l_1)$, because $p_i \leq y$ (see Figure 9). Therefore, S^l_2 can be replaced by S in $S^l_{1,2}$, and $|\mathcal{J}(S)| > |\mathcal{J}(S^l_2)|$. This leads to a contradiction with Property 1 on S^l_2. □

In order to prove Proposition 2, i.e. that $|\mathcal{J}(S^{D_2})| \leq |\mathcal{J}(S^l_{1,2})| + 1$, we will proceed by contradiction, and suppose, from now on, that $|\mathcal{J}(S^{D_2})| \geq |\mathcal{J}(S^l_{1,2})| + 2$. We introduce now some notations, before introducing partial schedule S^{mid}.

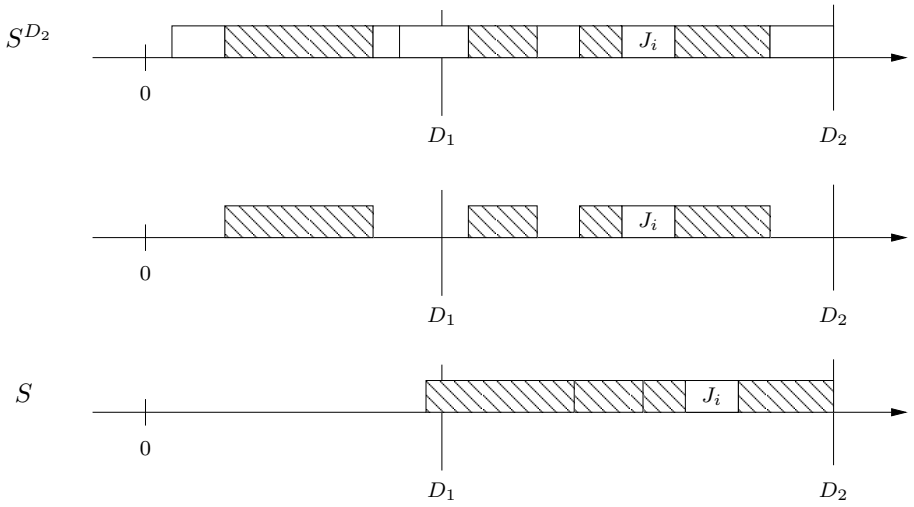
NOTATIONS. Let $\bar{l}$ be the number of jobs of $\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_{1,2}) = \{J_{e_1}, \ldots, J_{e_{\bar{l}}}\}$ ($J_{e_1} \prec_{ERD} \ldots \prec_{ERD} J_{e_{\bar{l}}}$, i.e. $e_1 < \ldots < e_{\bar{l}}$). Let $\overline{d}$ be the number of jobs of $\mathcal{J}(S^l_{1,2})\backslash\mathcal{J}(S^{D_2})$.

By Lemma 2, $\overline{d} = \mathcal{J}(S^l_1)\backslash\mathcal{J}(S^{D_2})$. And since $|\mathcal{J}(S^{D_2})| \geq |\mathcal{J}(S^l_{1,2})| + 2$, we have: $0 \leq \overline{d} \leq \bar{l} - 2$.

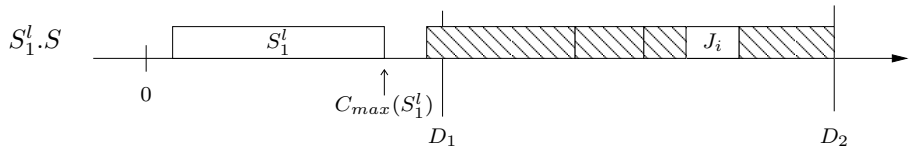
NOTATIONS. Let $l_2 = |\mathcal{J}(S^l_2)|$, and $J_{f_1}, \ldots, J_{f_{l_2}}$ be the jobs of S^l_2 ($J_{f_1} \prec_{ERD} \ldots \prec_{ERD} J_{f_{l_2}}$).



(Partial) schedule $S^l_{1,2}$, and job J_i, which is not in the schedule, and whose processing time is shorter than y.



From top to bottom: (partial) schedule S^{D_2}, an intermediate partial schedule, partial schedule S.



(Partial) schedule $S^l_1.S$.

Fig. 9: Illustrating the proof of Lemma 3.

From Lemma 2, $J_{f_1}, \ldots, J_{f_{l_2}} \in \mathcal{J}(S^{D_2})$. On Figure 10 an example is given, where $\overline{l} = 6$, $\overline{d} = 4$ and $l_2 = 5$. The same example will be used in the rest of the proof to illustrate it.

$\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_{1,2})$ and $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_{1,2})$ constitute a partition of $\mathcal{J}(S^{D_2})$. Moreover, since $\mathcal{J}(S^l_1)$ and $\mathcal{J}(S^l_2)$ constitute a partition of $\mathcal{J}(S^l_{1,2})$, $\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_1)$ and $\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_2)$ constitute a partition of $\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_{1,2})$. Therefore, a partition of $\mathcal{J}(S^{D_2})$ is constituted of the three sets of jobs:

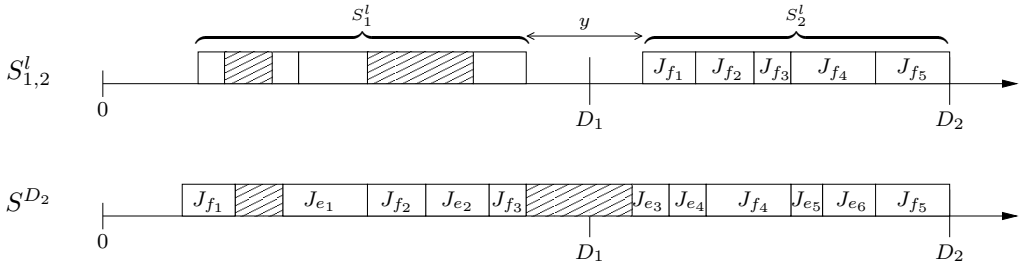


Fig. 10: On top, the schedule represents $S^l_{1,2}$, while the schedule at bottom represents S^{D_2}, for the same instance. In both schedules, the striped jobs belong to $\mathcal{J}(S^l_1) \cap \mathcal{J}(S^{D_2})$. In S^l_1, the white jobs belong to $\mathcal{J}(S^l_1) \backslash \mathcal{J}(S^{D_2})$. On this example, we have: $\overline{l} = 6$ (the number of J_{e_i} jobs), $\overline{d} = 4$ (the number of white jobs in S^l_1), $l_2 = 5$ (the number of J_{f_j} jobs).

- $\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_1)$,
- $\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S^l_2) = \mathcal{J}(S^l_2) = \{J_{f_1}, \ldots, J_{f_{l_2}}\}$,
- $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_{1,2}) = \{J_{e_1}, \ldots, J_{e_{\overline{l}}}\}$

Then, $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_1) = \{J_{f_1}, \ldots, J_{f_{l_2}}\} \cup \{J_{e_1}, \ldots, J_{e_{\overline{l}}}\}$. Hence, $|\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_1)| = \overline{l} + l_2 \geq \overline{d} + 2$.

NOTATION. Let G be the set of the $\overline{d}+1$ first jobs of $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_1)$ in ERD order.

Equivalently, G is also the subset of jobs of $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_1)$ with the $\overline{d}+1$ smallest indices. Moreover, the jobs of G are also the $\overline{d}+1$ earliest scheduled jobs of $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_1)$ in S^{D_2}, from Property 3 on S^{D_2}.

On the example of Figure 10, there are $\overline{d}+1 = 5$ jobs in G: $J_{f_1}, J_{e_1}, J_{f_2}, J_{e_2}, J_{f_3}$.

NOTATION. Let $G^e = G \cap \{J_{e_1}, \ldots, J_{e_{\overline{l}}}\}$ and $G^f = G \cap \{J_{f_1}, \ldots, J_{f_{l_2}}\}$.

G^e and G^f constitute a partition of G, since $G \subseteq \mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_1)$, and the two sets $\{J_{e_1}, \ldots, J_{e_{\overline{l}}}\}$ and $\{J_{f_1}, \ldots, J_{f_{l_2}}\}$ are a partition of $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_1)$. Therefore, $|G^e| + |G^f| = |G| = \overline{d} + 1$.

Since the jobs of G have the smallest indices among the jobs of $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_1)$, we deduce that $\{J_{e_1}, \ldots, J_{e_{\overline{l}}}\} \cap G = \{J_{e_1}, \ldots, J_{e_{|G^e|}}\}$ (since $e_1 < \ldots < e_{|G^e|} < \ldots < e_{\overline{l}}$), and $\{J_{f_1}, \ldots, J_{f_{l_2}}\} \cap G = \{J_{f_1}, \ldots, J_{f_{|G^f|}}\}$ (since $f_1 < \ldots < f_{|G^f|} < \ldots < f_{l_2}$).

On the example of Figure 10, $G^e = \{J_{e_1}, J_{e_2}\}$ and $G^f = \{J_{f_1}, J_{f_2}, J_{f_3}\}$.

We introduce now a new partial schedule S^{mid} that is necessary for the proof of Proposition 2.

Partial schedule S^{mid}

S^{mid} is an intermediate feasible partial schedule between $S^l_{1,2}$ and S^{D_2}: S^{mid} contains all the jobs of $\mathcal{J}(S^l_{1,2}) \cap \mathcal{J}(S^{D_2})$ and some other jobs of S^{D_2}. By the means of S^{mid}, we show that, if $|\mathcal{J}(S^{D_2})| \geq |\mathcal{J}(S^l_{1,2})| + 2$, then there must be a job of $\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S^l_{1,2})$ whose processing time is shorter than $y = B(S^l_2) - C_{max}(S^l_1)$. Thus we find a contradiction with Lemma 3.

We first show how S^{mid} is constructed, then prove some properties of S^{mid} and S^{D_2}, and finally show that S^{mid} is feasible.

The partial schedule S^{mid} is constructed in the following way, starting from $S^l_{1,2}$ (see Figure 11). S^{mid} is composed of two subschedules S_1^{mid} and S_2^{mid}. S_1^{mid} is a left-shifted subschedule obtained from S_1^l by removing the $\overline{d}$ jobs of $\mathcal{J}(S_1^l)\backslash\mathcal{J}(S^{D_2})$, and by adding the $\overline{d}+1$ jobs of G to S_1^l, while maintaining the order of increasing indices (ERD order) guaranteed by Property 3 in S_1^l. S_2^{mid} is the schedule obtained by removing $J_{f_1},\ldots,J_{f_{|G^f|}}$ from S_2^l.

Remark: Until now notation S_1 designed a subschedule of a schedule S, where all the jobs complete at or before D_1. However, for S^{mid}, we use this notation to design the first (left-shifted) part of the schedule, even if some jobs of S_1^{mid} complete after D_1.

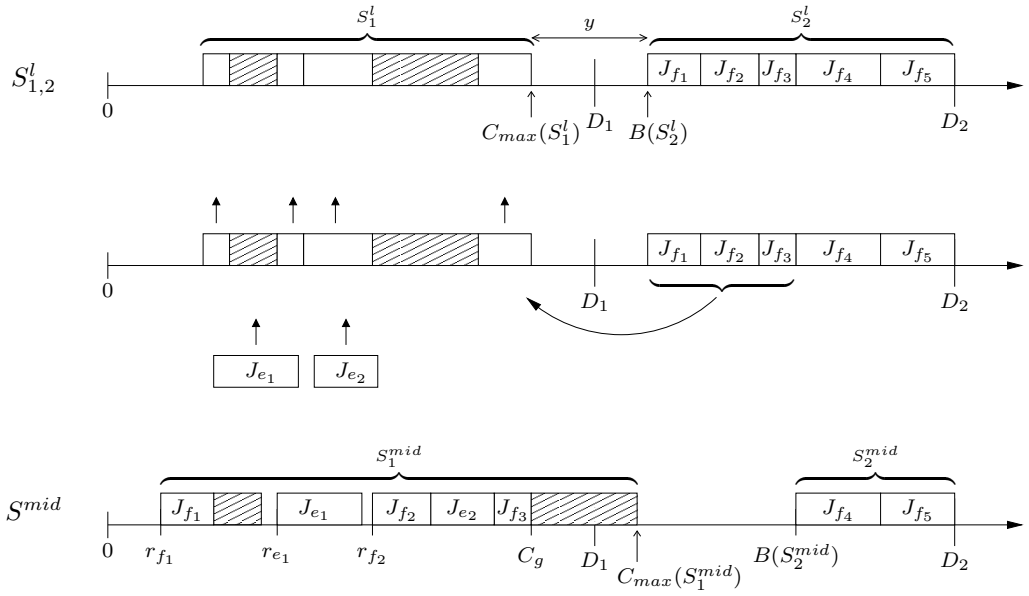


Fig. 11: Starting from $S^l_{1,2}$, we obtain S^{mid}: the $\overline{d}=4$ jobs of $\mathcal{J}(S_1^l)\backslash\mathcal{J}(S^{D_2})$ are removed from S_1^l; the $|G^e|=2$ jobs of $G\cap(\mathcal{J}(S^{D_2})\backslash\mathcal{J}(S^l_{1,2}))$ are added to S_1^l, and the $|G^f|=3$ jobs of $\mathcal{J}(S_2^l)\cap G$ are moved from S_2^l to S_1^{mid}.

We now introduce some properties of S^{mid}.

Feature 1 *S_1^{mid} is left-shifted.*

Proof By construction. □

Feature 2 *S_2^{mid} is a right-shifted block that completes at D_2.*

Proof S_2^{mid} is the second part of the schedule S_2^l, which is also a block that completes at D_2. More precisely, S_2^{mid} is the block that corresponds to the sequence $(J_{f_{|G^f|+1}},\ldots,J_{f_{l_2}})$ and that completes at D_2 (see Figure 11). □

Feature 3 $|\mathcal{J}(S_1^{mid})| = |\mathcal{J}(S_1^l)| + 1$.

Proof By construction: to obtain S_1^{mid}, $\overline{d}$ jobs were removed from S_1^l, while $\overline{d}+1$ were added (i.e. the $\overline{d}+1$ first jobs of G). □

Feature 4 $\mathcal{J}(S^{mid}) \subseteq \mathcal{J}(S^{D_2})$.

Proof By construction: to construct S^{mid} we removed from $S_{1,2}^l$ the jobs that were not in S^{D_2} and we added the jobs of $G \subseteq \mathcal{J}(S^{D_2})$. □

Feature 5 $C_{max}(S_1^{mid}) > D_1$.

Proof By contradiction, assume that $C_{max}(S_1^{mid}) \leq D_1$: since $|\mathcal{J}(S_1^{mid})| = |\mathcal{J}(S_1^l)|+1$ (cf. Feature 3), it means that $|\mathcal{J}(S_1^l)|+1$ jobs of $\mathcal{J}^{all}$ can be scheduled in $[0, D_1]$, which is in contradiction with Property 1 on S_1^l. □

NOTATION. Let J_g be the job of G that has the greatest index (i.e. the last job of G in ERD order). It is also the last job of G being scheduled in S_1^{mid} (see Figure 11, where $J_g = J_{f_3}$).

Feature 6 *The jobs that are scheduled later than* J_g *in* S_1^{mid} *all belong to* $\mathcal{J}(S_1^l)$*:* $(\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})) \subseteq \mathcal{J}(S_1^l)$.

Proof By construction of S_1^{mid}, the only jobs of S_1^{mid} that do not belong to S_1^l, are the jobs of G. Since the last scheduled job of G in S_1^{mid} is J_g (by definition of J_g), the jobs that are scheduled later in S_1^{mid} all belong to S_1^l. Moreover, since by construction the jobs in S_1^{mid} are ordered by increasing indices, $\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})$ is actually the set of jobs scheduled after J_g in S_1^{mid}. □

NOTATION. Let $\Gamma_g^{D_2}$ be the sequence of jobs that are scheduled between 0 and $C_g(S^{D_2})$ in S^{D_2}; and Γ_g^{mid} the sequence of jobs that are scheduled between 0 and $C_g(S_1^{mid})$ in S_1^{mid}.

Feature 7 $\Gamma_g^{mid} = \Gamma_g^{D_2}$.

Proof First, note that $J_g \in G \subseteq \mathcal{J}(S^{D_2})$. From Property 3 on S^{D_2}, $\mathcal{J}(\Gamma_g^{D_2}) = (\mathcal{J}_g^- \cap \mathcal{J}(S^{D_2})) \cup \{J_g\}$. Moreover, since the jobs of S_1^{mid} are ordered by increasing indices (by construction), $\mathcal{J}(\Gamma_g^{mid}) = (\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^-) \cup \{J_g\}$. Thus, showing that $\mathcal{J}(\Gamma_g^{mid}) \cap \mathcal{J}_g^- = \mathcal{J}(\Gamma_g^{D_2}) \cap \mathcal{J}_g^-$ directly implies $\mathcal{J}(\Gamma_g^{mid}) = \mathcal{J}(\Gamma_g^{D_2})$.

By construction of S_1^{mid}, the two sets $\mathcal{J}(S_1^l) \cap \mathcal{J}(S^{D_2})$ and G constitute a partition of $\mathcal{J}(S_1^{mid})$. Therefore, $\mathcal{J}(S_1^l) \cap \mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^-$ and $G \cap \mathcal{J}_g^-$ constitute a partition of $\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^- = \mathcal{J}(\Gamma_g^{mid}) \cap \mathcal{J}_g^-$. Notice that these two sets are also a partition of $\mathcal{J}_g^- \cap \mathcal{J}(S^{D_2}) = \mathcal{J}_g^- \cap \mathcal{J}(\Gamma_g^{D_2})$, since $G \cap \mathcal{J}_g^- = (\mathcal{J}_g^- \cap \mathcal{J}(S^{D_2})) \backslash \mathcal{J}(S_1^l)$. Hence, $\mathcal{J}(\Gamma_g^{mid}) \cap \mathcal{J}_g^- = \mathcal{J}(\Gamma_g^{D_2}) \cap \mathcal{J}_g^-$.

Furthermore, since the jobs in S_1^{mid}, respectively S^{D_2}, are ordered w.r.t. increasing indices (by construction on S_1^{mid} and from Property 3 on S^{D_2}), we deduce that the jobs of $\Gamma_g^{D_2}$ and Γ_g^{mid} are in the same order. □

Feature 8 $\mathcal{J}(S_2^{mid}) \subseteq \mathcal{J}_g^+$.

Proof $\mathcal{J}(S^{mid}) \subseteq \mathcal{J}(S^{D_2})$ (cf. Feature 4). Then, since the jobs that precede J_g in S^{mid} are the same that the ones preceding J_g in S^{D_2} (cf. Feature 7), we deduce that the jobs that follow J_g in S^{mid} are a subset of the jobs that follow J_g in S^{D_2}. From Property 3 on S^{D_2}, the jobs that follow J_g in S^{D_2} are those of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})$. Then, since all the jobs of S_2^{mid} are scheduled after J_g in S^{mid} (because $J_g \in \mathcal{J}(S_1^{mid})$), we deduce that $\mathcal{J}(S_2^{mid}) \subseteq \mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2}) \subseteq \mathcal{J}_g^+$. □

Feature 9 *In S^{D_2}, all the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})$ are scheduled after $C_g(S_1^{mid})$.*

Proof From Property 3 on S^{D_2}, all the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})$ are scheduled after J_g.

Let us show that J_g cannot complete earlier than $C_g(S_1^{mid})$ in S^{D_2}. From Feature 7, the sequence of jobs scheduled in S_1^{mid} between 0 and $C_g(S_1^{mid})$ is the same as that of the jobs scheduled in S^{D_2} between 0 and $C_g(S^{D_2})$. Therefore, since S_1^{mid} is left-shifted (cf. Feature 1), J_g cannot complete earlier than $C_g(S_1^{mid})$ in S^{D_2}. Consequently, all the jobs that follow J_g in S^{D_2} are scheduled after $C_g(S_1^{mid})$ in S^{D_2}. □

Remark: In particular, the result of Feature 9 is true for any subset of jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})$, and thus for $\mathcal{J}_g^+ \cap \mathcal{J}(S^{mid})$ (cf. Feature 4): all the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{mid})$ are scheduled after $C_g(S_1^{mid})$ in S^{D_2}.

Feature 10 $p(\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})) \leq D_2 - C_g(S_1^{mid})$.

Proof Since, by Feature 9, all the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{D_2})$ are scheduled between $C_g(S_1^{mid})$ and D_2 in S^{D_2}, their total processing time must be less than or equal to $D_2 - C_g(S_1^{mid})$. □

Remark 1: In particular, the sum of the processing times of the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S^{mid})$ is also less than or equal to $D_2 - C_g(S_1^{mid})$.

Remark 2: However, notice that even though $\mathcal{J}(S^{mid}) \cap \mathcal{J}_g^+ \subseteq \mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+$, the sequence of the jobs of $\mathcal{J}(S^{mid}) \cap \mathcal{J}_g^+$ in S^{mid} is not a subsequence of the sequence of jobs of $\mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+$ in S^{D_2}. Indeed, the jobs scheduled after J_g in S^{D_2} are scheduled in the order of their increasing indices, while the jobs scheduled after J_g in S^{mid} are partitioned in two groups: the jobs of $\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+$, which are ordered by increasing indices, and the jobs of S_2^{mid}, which are also ordered by increasing indices; but there is no guarantee that the jobs of $(\mathcal{J}(S_1^{mid} \cap \mathcal{J}_g^+)) \cup \mathcal{J}(S_2^{mid})$ are scheduled in the increasing order of their indices in S^{mid}.

Feature 11 *In S_1^{mid}, there are no idle times between any pair of jobs of $\{J_g\} \cup (\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid}))$.*

Proof By contradiction, suppose that there is an idle time in S_1^{mid} between a pair of jobs of $\{J_g\} \cup (\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid}))$. Then, since S_1^{mid} is left-shifted (cf. Feature 1) there is at least one job J_q of $\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})$ that starts exactly at its release date r_q. Moreover, $\{J_q\} \cup (\mathcal{J}_q^+ \cap \mathcal{J}(S_1^{mid})) \subseteq \mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid}) \subseteq \mathcal{J}(S_1^l)$ (cf. Feature 6). Furthermore, the jobs of $(\mathcal{J}_q^+ \cap \mathcal{J}(S_1^{mid})) \cup \{J_q\}$ are ordered by increasing indices both in S_1^{mid} (by construction) and in S_1^l (by Property 3).

Then, since J_q cannot start earlier than r_q in S_1^l, and since S_1^{mid} is left-shifted (cf. Feature 1), we deduce that S_1^l cannot complete before $C_{max}(S_1^{mid}) > D_1$ (cf. Feature 5), which is clearly a contradiction with the definition of S_1^l (a partial schedule that schedules jobs between 0 and D_1). □

Let us now show that S^{mid} is feasible.

Proposition 3 *S^{mid} is feasible.*

Proof By construction, each job starts at or after its release date in S^{mid}. Therefore, schedule S^{mid} is unfeasible if and only if $C_{max}(S_1^{mid}) > B(S_2^{mid})$. Let us prove by contradiction that S^{mid} is feasible. We assume that $C_{max}(S_1^{mid}) > B(S_2^{mid})$.

Therefore, if we examine S^{mid} between $C_g(S_1^{mid})$ and D_2 (see Figure 12), we see that the jobs of $\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})$ are left-shifted, starting from $C_g(S_1^{mid})$ in a unique block (cf. Feature 11), hence $C_{max}(S_1^{mid}) = C_g(S^{mid}) + p(\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid}))$. On the other hand, the jobs of S_2^{mid} are right-shifted, completing at D_2 in a unique block (cf. Feature 2), hence $B(S_2^{mid}) = D_2 - p(\mathcal{J}(S_2^{mid}))$.

$C_{max}(S_1^{mid}) > B(S_2^{mid})$ implies that $p((\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})) \cup \mathcal{J}(S_2^{mid})) = p(\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid})) + p(\mathcal{J}(S_2^{mid})) > D_2 - C_g(S_1^{mid})$, which is in contradiction with Feature 10, since $\mathcal{J}(S^{mid}) \subseteq \mathcal{J}(S^{D_2})$ (cf. Feature 4), $\mathcal{J}_g^+ \cap \mathcal{J}(S_1^{mid}) \subseteq \mathcal{J}_g^+$ and $\mathcal{J}(S_2^{mid}) \subseteq \mathcal{J}_g^+$ (cf. Feature 8). Consequently, S^{mid} is feasible. □

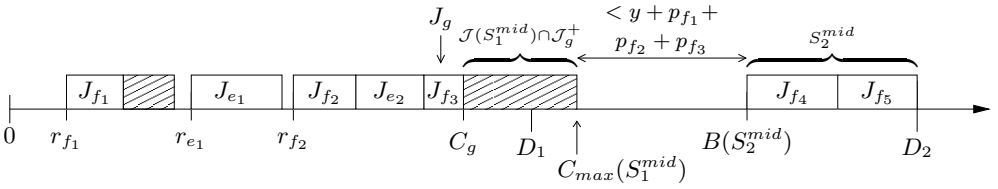


Fig. 12: Illustration of schedule S^{mid}.

We can now prove **Proposition 2** ($|\mathcal{J}(S^{D_2})| \leq |\mathcal{J}(S_{1,2}^l)| + 1$).

Proof We have the following partitions of $\mathcal{J}(S^{D_2})$:
$\mathcal{J}(S^{D_2}) = (\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S_{1,2}^l)) \cup (\mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S_{1,2}^l))$
$= (\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S_{1,2}^l)) \cup \{J_{e_1}, \ldots, J_{e_{\overline{l}}}\}$
$= (\mathcal{J}(S^{D_2}) \cap \mathcal{J}(S_{1,2}^l)) \cup G^e \cup \{J_{e_{|G^e|+1}}, \ldots, J_{e_{\overline{l}}}\}$
$= \mathcal{J}(S^{mid}) \cup \{J_{e_{|G^e|+1}}, \ldots, J_{e_{\overline{l}}}\}$

Hence, $\mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+ = (\mathcal{J}(S^{mid}) \cap \mathcal{J}_g^+) \cup \{J_{e_{|G^e|+1}}, \ldots, J_{e_{\overline{l}}}\}$, since $\{J_{e_{|G^e|+1}}, \ldots, J_{e_{\overline{l}}}\} \subseteq \mathcal{J}_g^+$, since $g < e_{|G^e|+1}$, by construction of G and by definition of g.

Notice that $\mathcal{J}(S^{mid}) \cap \mathcal{J}_g^+ = (\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+) \cup \mathcal{J}(S_2^{mid})$, since $\mathcal{J}(S_2^{mid}) \subseteq \mathcal{J}_g^+$ (cf. Feature 8). Then, since the three sets $\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+$, $\mathcal{J}(S_2^{mid})$ and $\{J_{e_{|G^e|+1}}, \ldots, J_{e_{\overline{l}}}\}$ are disjoint, they constitute a partition of $\mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+$. Consequently, $p(\mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+) = p(\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+) + p(\mathcal{J}(S_2^{mid})) + p(\{J_{e_{|G^e|+1}}, \ldots, J_{e_{\overline{l}}}\}) \leq D_2 - C_g(S_1^{mid})$, since all the jobs of $\mathcal{J}(S^{D_2}) \cap \mathcal{J}_g^+$ are scheduled after $C_g(S_1^{mid})$ in S^{D_2} (cf. Feature 9). We have:

- $p(\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+) = C_{max}(S_1^{mid}) - C_g(S_1^{mid})$, since the jobs of $\mathcal{J}(S_1^{mid}) \cap \mathcal{J}_g^+$ are scheduled without idle times between $C_g(S_1^{mid})$ and $C_{max}(S_1^{mid})$ in S^{mid} (cf. Feature 11); and

– $p(\mathcal{J}(S_2^{mid})) = D_2 - B(S_2^{mid})$, since the jobs of $\mathcal{J}(S_2^{mid})$ are scheduled without idle times between $B(S_2^{mid})$ and D_2 in S^{mid} (cf. Feature 2).

We deduce that $p(\{J_{e_{|G^e|+1}}, \ldots, J_{e_{\bar{l}}}\}) \leq D_2 - C_g(S_1^{mid}) - (C_{max}(S_1^{mid}) - C_g(S_1^{mid})) - (D_2 - B(S_2^{mid})) = B(S_2^{mid}) - C_{max}(S_1^{mid})$; which can be written as: $\sum_{i=|G^e|+1}^{\bar{l}} p_{e_i} \leq B(S_2^{mid}) - C_{max}(S_1^{mid})$. Moreover, we have:

– $B(S_2^{mid}) - C_{max}(S_1^{mid}) < B(S_2^{mid}) - C_{max}(S_1^l)$ because $C_{max}(S_1^l) \leq D_1 < C_{max}(S_1^{mid})$ (cf. Feature 5); and
– $B(S_2^{mid}) = B(S_2^l) + \sum_{j=1}^{|G^f|} p_{f_j}$, by construction of S^{mid} (see Figure 11); and hence
– $B(S_2^{mid}) - C_{max}(S_1^l) = B(S_2^l) - C_{max}(S_1^l) + \sum_{j=1}^{|G^f|} p_{f_j} = y + \sum_{j=1}^{|G^f|} p_{f_j}$, by definition of y (see Figure 11).

From what precedes, $\sum_{i=|G^e|+1}^{\bar{l}} p_{e_i} < y + \sum_{j=1}^{|G^f|} p_{f_j}$, which can be written as:

$$\sum_{i=1}^{\bar{l}-|G^e|} p_{e_{i+|G^e|}} < y + \sum_{j=1}^{|G^f|} p_{f_j}$$

Notice that $\bar{l} - |G^e| > |G^f|$, since $|G^f| = |G| - |G^e| = \bar{d} + 1 - |G^e|$ and $\bar{d} \leq \bar{l} - 2$. Then, for all $i \in \{1, \ldots, |G^f|\}$:

– $J_{e_{i+|G^e|}} \in \mathcal{J}(S^{D_2}) \backslash \mathcal{J}(S_{1,2}^l)$
– $J_{f_i} \in \mathcal{J}(S_2^l)$, and
– $\mathcal{J}_{f_i}^+ \cap \mathcal{J}(S_2^l) \subseteq \mathcal{J}(S^{D_2})$ (since $\mathcal{J}(S_2^l) \subseteq \mathcal{J}(S^{D_2})$ by Lemma 2).

Therefore, by Lemma 1: $\forall i \in \{1, \ldots, |G^f|\}, p_{e_{i+|G^e|}} \geq p_{f_i}$. Hence, $\sum_{i=|G^f|+1}^{\bar{l}-|G^e|} p_{e_{i+|G^e|}} < y$; and this sum has at least one term, since $\bar{l} - |G^e| > |G^f|$. Therefore, $p_{e_{\bar{l}}} < y$, while, from Lemma 3, $p_{e_{\bar{l}}} > y$, which is a contradiction. □

Hence, the result of Theorem 1 can be generalized to the following case.

Corollary 1 *Let us consider any pair of values v_1, v_2 such that $v_1 > v_2$, and the job stepwise payoff function equal to v_1 (resp. v_2, resp. 0) on $[0, D_1]$ (resp. $]D_1, D_2]$, resp. $]D_2, D_3]$). Then, solving the single machine scheduling problem with unequal release dates and the maximization of the jobs' payoff function with algorithm APPROX yields a solution whose payoff v is such that $v \geq OPT - v_2$.*

Proof Proposition 2 shows that algorithm APPROX yields a solution where U_1 jobs complete at or before D_1, and at least $U_2 - 1$ jobs complete at or before D_2. Therefore its payoff is at least $(v_1 - v_2) \times U_1 + v_2 \times (U_2 - 1)$, while the upper bound on an optimal payoff is $(v_1 - v_2) \times U_1 + v_2 \times U_2$. Hence the result holds. □

5 Conclusion

We have shown a polynomial time algorithm with the best possible absolute guarantee (equal to 1) for the single machine scheduling problem with cumulative payoffs related to two delivery dates, $1|r_i|V_1 + V_2$.

Further research will be devoted to find a similar result for the general problem with K delivery dates: $1|r_i|\sum_{k=1}^{K} V_k$. We conjecture that for this problem algorithm APPROX (extended) yields a solution with an absolute performance guarantee of $K(K-1)/2$.

Acknowledgements This work was supported by FUI project Dem@tFactory, financed by DGCIS, Conseil général de Seine-et-Marne, Conseil général de Seine-Saint-Denis, Ville de Paris, Pôle de compétitivité Cap Digital.

References

1. Curry, J. and Peters, B. (2005). Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stabimity objectives. *International Journal of Production Research, 43(15)*, 3231–3246.
2. Detienne, B., Dauzères-Pérès, S. and Yugma, C. (2011). Scheduling jobs on parallel machines to minimize a regular step total cost function. *Journal of Scheduling, 14(6)*, 523–538.
3. Detienne, B., Dauzères-Pérès, S., and Yugma, C. (2012). An exact approach for scheduling jobs with regular step cost functions on a single machine. *Computers & Operations Research, 39(5)*, 1033–1043.
4. Graham, R. L., Lawler, E. L., Lenstra, J. K. and Rinnooy Kann, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey, *Annals of Discrete Mathematics, 6*, 287–326.
5. Hall, N. G., Lesaoana, M. and Potts, C. N. (2001). Scheduling with Fixed Delivery Dates. *Operations Research, 49(1)*, 134–144 .
6. Hall, N. G., Sethi, S. P. and Sriskandarajah,C. (1991). On the complexity of generalized due date scheduling problems. *European Journal of Operational Research, 51(1)*, 100–109.
7. Janiak, A. and Krysiak, T. (2007). Single processor scheduling with job values depending on their completion times. *Journal of Scheduling, 10(2)*, 129–138.
8. Janiak, A. and Krysiak, T. (2012). Scheduling jobs with values dependent on their completion times. *International Journal of Production Economics, 135(1)*, 231–241.
9. Moore, J. M. (1968). An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs. *Management Science, 15(1)*, 102–109.
10. Sahin, G. and Ahuja, R. K. (2011). Single-machine scheduling with stepwise tardiness costs and release times. *Journal of Industrial Management and Optimization, 7(4)*, 825–848.
11. Y. Seddik, C. Gonzales and S. Kedad-Sidhoum (2013). Single machine scheduling with delivery dates and cumulative payoffs. *Journal of Scheduling, 16(3)*, 313–329.
12. Y. Seddik (2012). Scheduling with delivery dates and cumulative payoffs. *PhD thesis, under the supervision of C. Gonzales and S. Kedad-Sidhoum.*
13. Tseng, C.-T., Chou, Y.-C. and Chen, W.-Y. (2010). A variable neighborhood search for the single machine total stepwise tardiness problem. *Proceedings of the 2010 International Conference on Engineering, Project and Production Management*, 101–108.
14. Yang, W.-H. (2009). Sequencing jobs subject to multiple common due dates. *Proceedings of the 10th Asia Pacific Industrial Engineering & Management Systems Conference*, 1041–1050.

MISTA 2013

A Graphical Algorithm for Solving an Investment Optimization Problem

Evgeny R. Gafarov · Alexandre Dolgui · Alexander A. Lazarev · Frank Werner.

Abstract In this paper, a graphical algorithm (GrA) is presented for an investment optimization problem. This algorithm is based on the same Bellman equations as the best known dynamic programming algorithm (DPA) for the problem but the GrA has several advantages in comparison with the DPA. Based on this GrA, a fully-polynomial time approximation scheme is proposed having the best known running time. The idea of the GrA presented can also be used to solve some similar scheduling or lot-sizing problems in a more effective way.

1 Introduction

The Project Investment Problem can be formulated as follows. A set N of n potential projects and an investment budget (amount) $A > 0,\ A \in Z$, are given. For each project $j,\ j = 1, \ldots, n$, a profit function $f_j(x),\ x \in [0, A]$, is given, where the value $f_j(x')$ denotes the profit received if the amount x' is invested into the project j. The objective is to determine an amount $x_j \in [0, A],\ x_j \in Z$, for each project $j \in N$ such that $\sum_{j:=1}^{n} x_j \leq A$ and the total profit $\sum_{j:=1}^{n} f_j(x_j)$ is maximized.

E.R. Gafarov
Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya st. 65, 117997 Moscow, Russia
E-mail: axel73@mail.ru

A. Dolgui
Ecole Nationale Superieure des Mines, CNRS UMR6158, LIMOS, F-42023 Saint-Etienne, France
E-mail: dolgui@emse.fr

A.A. Lazarev
Lomonosov Moscow State University, Higher School of Economics (National Research University), Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya st. 65, 117997 Moscow, Russia,
E-mail: jobmath@mail.ru

F. Werner
Fakultät für Mathematik, Otto-von-Guericke-Universität Magdeburg, PSF 4120, 39016 Magdeburg, Germany,
E-mail: frank.werner@ovgu.de

In this paper, we deal with piecewise linear functions $f_j(x)$. Suppose that the interval $[0, A]$ can be written as

$$[0, A] = [t_j^0, t_j^1] \bigcup (t_j^1, t_j^2] \bigcup \ldots \bigcup (t_j^{k-1}, t_j^k] \bigcup \ldots \bigcup (t_j^{k_j-1}, t_j^{k_j}]$$

such that the profit function has the form $f_j(x) = b_j^k + u_j^k(x - t_j^{k-1})$, if $x \in (t_j^{k-1}, t_j^k]$, where k is the number of the interval, b_k^j is the value of the function at the beginning of the interval, and u_j^k is the slope of the function. Without loss of generality, assume that $b_j^1 \leq b_j^2 \leq \ldots \leq b_j^{k_j}$ and $t_j^k \in Z,\ j \in N,\ k = 1, 2, \ldots, k_j$, and that $t_j^{k_j} = A,\ j = 1, 2, \ldots, n$.

A special case of this problem is similar to the well-known bounded knapsack problem:

$$\begin{array}{ll} \text{maximize} & \sum_{j:=1}^{n} p_j x_j \\ s.t. & \sum_{j:=1}^{n} w_j x_j \leq A, \\ & x_j \in [0, b_j],\ x_j \in Z,\ j = 1, 2, \ldots, n, \end{array} \tag{1}$$

for which a dynamic programming algorithm (DPA) of time complexity $O(nA)$ is known [3].

The following problem is also similar to the problem under consideration:

$$\begin{array}{ll} \text{minimize} & \sum_{j:=1}^{n} f_j(x_j) \\ s.t. & \sum_{j:=1}^{n} x_j \geq A, \\ & x_j \in [0, A],\ x_j \in Z,\ j = 1, 2, \ldots, n, \end{array} \tag{2}$$

where $f_j(x_j)$ are piecewise linear as well. For this problem, a DPA with a running time of $O(\sum k_j A)$ [4] and a fully polynomial-time approximation scheme (FPTAS) with a running time of $O((\sum k_j)^3/\varepsilon)$ [5] are known.

In this paper, we present an alternative solution algorithm with a running time of $O(\sum k_j A)$ and an FPTAS based on this solution algorithm with a running time of $O(\sum k_j n \log\log n/\varepsilon)$.

The remainder of the paper is as follows. In Section 2, we present the Bellman equations to solve the problem under consideration. In Section 3, a graphical algorithm (GrA) based on an idea from [1] is presented. In Section 4, an FPTAS based on this GrA is derived.

2 Dynamic programming algorithm

In this section, we present a DPA for the problem considered. For any project j and any state $t \in [0, A]$, we define $F_j(t)$ as the maximal profit incurred for the projects $1, 2, \ldots, j$, when the remaining budget available for the projects $j + 1, j + 2, \ldots, n$ is equal to t. Thus, we have:

$$\begin{array}{ll} F_j(t) = \max & \sum_{h:=1}^{j} f_h(x_h) \\ s.t. & \sum_{h:=1}^{j} x_h \leq A - t, \\ & x_h \geq 0,\ x_h \in Z,\ h = 1, 2, \ldots, j. \end{array} \tag{3}$$

We define $F_j(t) = 0$ for $t \notin [0, A]$. Then we have the following recursive equations:

$$F_j(t) = \max_{x\in[0,A-t]}\{f_j(x) + F_{j-1}(t+x)\}$$
$$= \max_{1\le k\le k_j} \max_{x\in(t_j^{k-1},t_j^k]\bigcap[0,A-t]} \{b_j^k - u_j^k t_j^{k-1} + u_j^k \cdot x + F_{j-1}(t+x)\}, \quad (4)$$
$$j = 1, 2, \ldots, n.$$

Lemma 1 *All functions $F_j(t),\ j = 1, 2, \ldots, n$, are non-increasing on the interval $[0, A]$.*

The proof of this lemma immediately follows from the definition of the functions $F_j(t)$.

The running time of the DPA using such a type of Bellman equations is $O(\sum k_j A)$ if we use an idea from [4].

3 Graphical algorithm

In this section, we present a GrA which constructs the functions $F_j(t),\ j = 1, 2, \ldots, n$, in a more effective way in comparison with the DPA. Below we prove that the functions $F_j(t),\ j = 1, 2, \ldots, n$, constructed in the GrA are piecewise linear (see Lemma 4).

Any piecewise linear function $\varphi(x)$ considered in this paper can be defined by three sets of numbers: a set of break points I (at each break point, a new linear segment of the piecewise linear function begins), a set of slopes U and a set of values of the function at the beginning of the interval B. Let the notation $I[k]$ denote the k-th element in the ordered set I. The same notations will be used for the sets U and B as well. The notation $\varphi.I[k]$ denotes the k-th element of the set I of the function $\varphi(x)$. Then, for example, for $x \in (t_j^{k-1}, t_j^k] = (f_j.I[k-1], f_j.I[k]]$, we have

$$f_j(x) = f_j.B[k] + f_j.U[k](x - f_j.I[k-1]).$$

Note that

$$\varphi.I[k] < \varphi.I[k+1],\ k = 1, \ldots, |\varphi.I| - 1 \quad \text{and} \quad k_j = |f_j.I|.$$

In each step $j,\ j = 1, 2, \ldots, n$, of the subsequent algorithm, the temporary piecewise linear functions Ψ_j^i and Φ_j^i are constructed. Recall that the functions $F_j(t),\ j = 1, 2, \ldots, n$, constructed in the GrA are piecewise linear as well. For $t \in Z$, their values are equal to the values of the functions $F_j(t)$ considered in the DPA.

Let $\varphi.I[-1] = 0$ and $\varphi.I[|\varphi.I| + 1] = A$. Remind that $\varphi.I[|\varphi.I|] = A$.
The points $t \in \varphi.I$ and the other end points of the intervals with the piecewise linear functions considered in this article will be called *break points*. To *construct a function* in the GrA means to compute their sets I, U and B.

Graphical algorithm

1. Let $F_0(t) = 0$, i.e., $F_0.I := \{A\},\ F_0.U := \{0\},\ F_0.B := \{0\}$;
2. FOR $j := 1$ TO n DO
   2.1. FOR $k := 1$ TO k_j DO
      2.1.1. Construct the temporary function

$$\Psi_j^k(t) = f_j.B[k] - f_j.U[k] \cdot f_j.I[k-1] + f_j.U[k] \cdot t + F_{j-1}(t)$$

   according to Procedure 2.1.1.;

2.1.2. Construct the temporary function

$$\Phi_j^k(t) = \max_{x \in (f_j.I[k-1], f_j.I[k]] \bigcap [0, A-t]} \{\Psi_j^k(t+x) - f_j.U[k] \cdot t\}$$

according to Procedure 2.1.2.;

2.1.3. IF $k = 1$ THEN $F_j^k(t) := \Phi_j^k(t)$ ELSE $F_j^k(t) := \max\{F_j^{k-1}(t), \Phi_j^k(t)\}$.

2.2. $F_j(t) := F_j^{k_j}(t)$. Modify the sets I, U, B of the function $F_j(t)$ according to Procedure 2.2.

3. The optimal objective function value is equal to $F_n(0)$.

Procedure 2.1.1.

Given are k and j;
$\Psi_j^k.I = \emptyset$, $\Psi_j^k.U = \emptyset$ and $\Psi_j^k.B = \emptyset$.
FOR $i := 1$ TO $|F_{j-1}.I|$ DO
add the value $F_{j-1}.I[i]$ to the set $\Psi_j^k.I$;
add the value

$$f_j.B[k] - f_j.U[k] \cdot f_j.I[k-1] + f_j.U[k] \cdot F_{j-1}.I[i] + F_{j-1}.B[i]$$

to the set $\Psi_j^k.B$;
add the value $f_j.U[k] + F_{j-1}.U[i]$ to the set $\Psi_j^k.U$;

In Procedure 2.1.1., we shift the function $F_{j-1}(t)$ up by the value $f_j.B[k] - f_j.U[k] \cdot f_j.I[k-1]$ and increase all slopes in its diagram by $f_j.U[k]$. If all values $t \in F_{j-1}.I$ are integer, then all values from the set $\Psi_j^i.I$ are integer as well. It is obvious that Procedure 2.1.1. can be performed in $O(|F_{j-1}.I|)$ time.

Before describing Procedure 2.1.2., we present Procedure FindMax in which the maximum function $\varphi(t)$ of two linear fragments $\varphi_1(t)$ and $\varphi_2(t)$ is constructed.

Procedure FindMax

1. Given are the functions $\varphi_1(t) = b_1 + u_1 \cdot t$ and $\varphi_2(t) = b_2 + u_2 \cdot t$ and an interval $(t', t'']$. Let $u_1 \leq u_2$;
2. IF $t'' - t' \leq 1$ THEN RETURN $\varphi(t) = \max\{\varphi_1(t''), \varphi_2(t'')\}$ defined on the interval $(t', t'']$;
3. Find the intersection point t^* of $\varphi_1(t)$ and $\varphi_2(t)$;
4. IF t^* does not exist OR $t^* \notin (t', t'']$ THEN
   IF $b_1 + u_1 \cdot t' > b_2 + u_2 \cdot t'$ THEN RETURN $\varphi(t) = \varphi_1(t)$ defined on the interval $(t', t'']$;
   ELSE RETURN $\varphi(t) = \varphi_2(t)$ defined on the interval $(t', t'']$;
5. ELSE
   IF $t^* \in Z$ THEN
      $\varphi(t) := \varphi_1(t)$ on the interval $(t', t^*]$;
      $\varphi(t) := \varphi_2(t)$ on the interval $(t^*, t'']$;
      RETURN $\varphi(t)$;
   ELSE IF $t^* \notin Z$ THEN
      $\varphi(t) := \varphi_1(t)$ on the interval $(t', \lfloor t^* \rfloor]$;
      $\varphi(t) := b_2 + u_2 \cdot \lfloor t^* \rfloor$ on the interval $(\lfloor t^* \rfloor - 1, \lfloor t^* \rfloor]$;
      $\varphi(t) := \varphi_2(t)$ on the interval $(\lfloor t^* \rfloor, t'']$;

RETURN $\varphi(t)$;

If both points t' and t'' are integer, then $\varphi.I$ contains only integer break points t. The running time of Procedure FindMax is constant.

In Procedure 2.1.2., we do the following. When we shift s' to the right, we shift the interval $T' = [t_{left}, t_{right}]$ of the length $f_j.I[k] - f_j.I[k-1]$. We have to use the values $\Psi_j^k(x)$ for $x \in T'$ to calculate $\Phi_j^k(t)$ at the point $t = s'$. Since $\Psi_j^k(x)$ is piecewise linear, it is only necessary to consider the values $\Psi_j^k(x)$ at the break points belonging to T' and at the end points of the interval T'. So, if we shift s' to the right by a small value $x \in [0, \varepsilon]$ such that all the break points remain the same, then the value $\Phi_j^k(t)$ will be changed according to the value $\varphi_{max}(x)$.

So, in cycle [2.1.2.5], we shift s' to the right. In steps [2.1.2.6]-[2.1.2.8], we look for the first and the last break point belonging to the current interval T'. For these two break points found and the best found break point with an index $s \in \{v, v+1, \ldots, w\}$ (step [2.1.2.14]), we construct the functions $\varphi_{left}(x), \varphi_{right}(x), \varphi_{inner}(x)$ and their maximum function $\varphi_{max}(x)$ (see steps [2.1.2.15]-[2.1.2.18]) according to which the value $\Phi_j^k(t)$ is changed when we shift s' to the right by a small value $x \in [0, \varepsilon]$. In steps [2.1.2.19]-[2.1.2.23], we construct the function $\Phi_j^k(t)$ based on the function $\varphi_{max}(x)$.

Procedure 2.1.2.

2.1.2.1. Given are k, j and $\Psi_j^k(t)$;
2.1.2.2. $\Phi_j^k.I := \emptyset$, $\Phi_j^k.U := \emptyset$ and $\Phi_j^k.B := \emptyset$;
2.1.2.3. $s' := 0$, $t_{left} := s' + f_j.I[k-1]$, $t_{right} := \min\{s' + f_j.I[k], A\}$;
2.1.2.4. Let $T' = \{\Psi_j^k.I[v], \Psi_j^k.I[v+1], \ldots, \Psi_j^k.I[w]\}$ be the maximal subset of $\Psi_j^k.I$, where $t_{left} < \Psi_j^k.I[v] < \ldots < \Psi_j^k.I[w] < t_{right}$,
Let $T := \{t_{left}\} \bigcup T' \bigcup \{t_{right}\}$;
2.1.2.5. WHILE $s' \leq A$ DO
2.1.2.6. IF $T' = \emptyset$ THEN let
$w+1 = argmax_{i=1,2,\ldots,|\Psi_j^k.I|}\{\Psi_j^k.I[i] | \Psi_j^k.I[i] > t_{right}\}$
and $v = argmax_{i=1,2,\ldots,|\Psi_j^k.I|}\{\Psi_j^k.I[i] | \Psi_j^k.I[i] > t_{left}\}$;
2.1.2.7. IF $w+1$ is not defined THEN let $w+1 = |\Psi_j^k.I|$;
2.1.2.8. IF v is not defined THEN let $v = |\Psi_j^k.I|$;
2.1.2.9. IF $t_{left} < A$ THEN $\varepsilon_{left} := \Psi_j^k.I[v] - t_{left}$ ELSE $\varepsilon_{left} := A - s'$;
2.1.2.10. IF $t_{right} < A$ THEN $\varepsilon_{right} := \Psi_j^k.I[w+1] - t_{right}$ ELSE $\varepsilon_{right} := +\infty$;
2.1.2.11. $\varepsilon := \min\{\varepsilon_{left}, \varepsilon_{right}\}$;
2.1.2.12. IF $t_{left} < A$ THEN

$$b_{left} := \Psi_j^k.B[v] + \Psi_j^k.U[v] \cdot (t_{left} - \Psi_j^k.I[v-1]) - f_j.U[k] \cdot s'$$

ELSE $b_{left} := 0$;
2.1.2.13. IF $t_{right} < A$ THEN

$$b_{right} := \Psi_j^k.B[w+1] + \Psi_j^k.U[w+1] \cdot (t_{right} - \Psi_j^k.I[w]) - f_j.U[k] \cdot s'$$

ELSE $b_{right} := 0$;
2.1.2.14. IF $T' = \emptyset$ THEN $b_{inner} := 0$ ELSE

$$b_{inner} := \max_{s=v,v+1,\ldots,w}\{\Psi_j^k.B[s] + \Psi_j^k.U[s] \cdot (\Psi_j^k.I[s] - \Psi_j^k.I[s-1])\} - f_j.U[k] \cdot s';$$

2.1.2.15. Denote function

$$\varphi_{left}(x) := b_{left} - (f_j.U[k] - \Psi_j^k.U[v]) \cdot x.$$

IF $t_{left} = A$ THEN $\varphi_{left}(x) := 0$;

2.1.2.16. Denote function

$$\varphi_{right}(x) := b_{right} - (f_j.U[k] - \Psi_j^k.U[w+1]) \cdot x.$$

IF $t_{right} = A$ THEN $\varphi_{right}(x) := 0$;

2.1.2.17. Denote function

$$\varphi_{inner}(x) := b_{inner} - f_j.U[k] \cdot x.$$

IF $T' = \emptyset$ THEN $\varphi_{inner}(x) := 0$;

2.1.2.18. Construct the piecewise linear function

$$\varphi_{max}(x) := \max_{x \in [0,\varepsilon]} \{\varphi_{left}(x), \varphi_{right}(x), \varphi_{inner}(x)\}$$

according to Procedure *FindMax*;

2.1.2.19. add the values from $\varphi_{max}.I$ increased by s' to the set $\Phi_j^k.I$;

2.1.2.20. add the values from $\varphi_{max}.B$ to the set $\Phi_j^k.B$;

2.1.2.21. add the values from $\varphi_{max}.U$ to the set $\Phi_j^k.U$;

2.1.2.22. IF $\varepsilon = \varepsilon_{left}$ THEN exclude $\Psi_j^k.I[v]$ from the set T and $v := v + 1$;

2.1.2.23. IF $\varepsilon = \varepsilon_{right}$ THEN include $\Psi_j^k.I[w+1]$ to the set T and $w := w + 1$;

2.1.2.24. $s' := s' + \varepsilon$.

2.1.2.25. $t_{left} := s' + f_j.I[k-1]$, $t_{right} := \min\{s' + f_j.I[k], A\}$;

2.1.2.26. Modify the function Φ_j^k according to Procedure 2.2.

Lemma 2 *Procedure 2.1.2. has a running time of* $O(|F_{j-1}^{k-1}.I|)$.

Proof. Step [2.1.2.14] has to be performed with the use of a simple data structure. Let $\{q_1, q_2, \ldots, q_r\}$ be a maximal subset of T' having the following properties:

$q_1 < q_2 < \ldots < q_r$;
there is no $j \in T'$ such that $q_i \leq j < q_{i+1}$ and
$\Psi_j^i.B[j] \geq \Psi_j^i.B[q_{i+1}]$, $i = 1, \ldots, r-1$.

We can keep track of the set $\{q_1, q_2, \ldots, q_r\}$ by storing its elements in increasing order in a Queue Stack, i.e., a list with the property that elements at the beginning can only be deleted while at the end, elements can be deleted and added [2]. This data structure can easily be implemented such that each deletion and each addition requires a constant time. So, step [2.1.2.14] can be performed in constant time.

Each of the steps [2.1.2.6]–[2.1.2.25] can be performed in constant time. The loop [2.1.2.5.] can be performed in $O(|\Psi_j^k.I|)$ time, where $|\Psi_j^k.I| = |F_{j-1}^{k-1}(t).I|$, since each time a break point from $|\Psi_j^k.I|$ is added or deleted. So, the lemma is true. □

We remind that in the DPA, the functional equations (4) are considered. In fact, in Procedure 2.1.1., we construct the function

$$b_j^k - u_j^k t_j^{k-1} + u_j^k \cdot (t + x) + F_{j-1}(t + x)$$

and in Procedure 2.1.2., we construct the function

$$\Phi_j^k(t) = \max_{x \in (t_j^{k-1}, t_j^k] \bigcap [0, A-t]} \{b_j^k - u_j^k t_j^{k-1} + u_j^k \cdot (t+x) - u_j^k \cdot t + F_{j-1}(t+x)\}.$$

Unlike the DPA, to construct $\Phi_j^k(t)$ in the GrA, we do not consider all integer points $x \in (t_j^{k-1}, t_j^k] \bigcap [0, A-t]$, but only the break points from the interval, since only they influence the values of $\Phi_j^k(t)$ (and in addition t_{left}, t_{right}). Step [2.1.3.] can be performed according to Procedure FindMax as well, i.e., to construct $F_j^k(t) := \max\{F_j^{k-1}(t), \Phi_j^i(t)\}$, linear fragments of $F_j^{k-1}(t)$ and $\Phi_j^i(t)$ have to be compared in each interval, organized by their break points. It is easy to see that we do the same operation with the integer points t as in the DPA. So, the values $F_j(t),\ t \in Z$, are the same for the GrA and the DPA, and we can state the following:

Lemma 3 *The values $F_j(t),\ j = 1, 2, \ldots, n$, at the points $t \in [0, A] \bigcap Z$ are equal to the values of the functions $F_j(t)$ considered in the DPA.*

Procedure 2.2.
Given is $F_j(t)$;
FOR $k := 1$ TO $|F_j.I| - 1$ DO

IF $F_j.U[k] = F_j.U[k+1]$ AND $F_j.U[k] \cdot (F_j.U[k] - F_j.U[k-1]) + F_j.B[k] = F_j.B[k+1]$ THEN
$F_j.B[k+1] := F_j.B[k]$;
Delete the kth elements from $F_j.B,\ F_j.U$ and $F_j.I$;

So, in Procedure 2.2., we combine two adjoining linear fragments that are in the same line. That means that, if we have two adjacent linear fragments which are described by the values (slopes) $F_j.U[k],\ F_j.U[k+1]$ and $F_j.B[k], F_j.B[k+1]$, where

$$F_j.U[k] \cdot (F_j.U[k] - F_j.U[k-1]) + F_j.B[k] = F_j.B[k+1],$$

(i.e., these fragments are on the same line), then, to reduce the number of intervals $|F_j.I|$ and thus the running time of the algorithm, we can join these two intervals into one interval.

Lemma 4 *All functions $F_j^i(t),\ j = 1, 2, \ldots, n,\ i = 1, 2, \ldots, k_j$, are piecewise linear on the interval $[0, A]$ with integer break points.*

Proof. It is obvious that function $F_0(t)$ is piecewise linear on the interval $[0, A]$. In Procedure 2.1.1., all break points from the set $\Psi_1^i.I$ are integer as well (see the comments after Procedure 2.1.1.). Since all points from $f_1.I$ are integer, we have $\varepsilon \in Z$ and as a consequence, $s' \in Z$. According to the Procedure FindMax, all points $\varphi_{max}.I$ considered in Procedure 2.1.2. are integer. So, all break points from $\Phi_1^i.I,\ i = 1, 2, \ldots, k_1$, are integer as well. Thus, the break points of the function $F_1^i(t) := \max\{F_1^{i-1}(t), \Phi_1^i(t)\}$ are integer, if we use Procedure FindMax to compute the function $\max\{F_1^{i-1}(t), \Phi_1^i(t)\}$. Analogously, we can prove that all break points of $F_2^i(t),\ i = 1, 2, \ldots, k_2$ are integer, etc.

Thus, it is obvious that all functions $F_j^i(t),\ j = 1, 2, \ldots, n,\ i = 1, 2, \ldots, k_j$, constructed in the GrA are piecewise linear. □

Theorem 1 *The GrA finds an optimal solution of the problem in*

$$O\left(\sum k_j \min\{A, \max_{j=1,2,\ldots,n} \{|F_j.B|\}\}\right)$$

time.

Proof. Analogously to the proof of Lemma 4, after each step [2.1.3.] of the GrA, the function $F_j^i(t),\ j = 1,2,\ldots,n,\ i = 1,2,\ldots,k_j$, has only integer break points from the interval $[0, A]$. Each function $\Phi_j^i.I,\ j = 1,2,\ldots,n, i = 1,2,\ldots,k_j$, has only integer break points from $[0, A]$ as well. So, to perform step [2.1.3.], we need to perform Procedure FindMax on no more than $A + 1$ intervals. Thus, the running time of step [2.1.3.] is $O(A)$. According to Lemmas 1 and 2, the running time of steps [2.1.1.] and [2.1.2.] is $O(F_j^i.I)$, where $F_j^i.I \leq A$. The running time of step [2.2.] is $O(F_j^i.I)$ as well.

Analogously to the comments after the DPA, it is easy to show that $F_j^i(t),\ j = 1,2,\ldots,n$, is a non-increasing function in t. Thus,

$$F_j^i.B[k] \geq F_j^i.B[k+1],\ j = 1,2,\ldots,n,\ k = 1,2,\ldots,|F_j^i.I| - 1.$$

Then, according to Procedure 2.2., there are no more than $2 \cdot F_j^i.B[0]$ different values in the set $F_j^i.I$, where $F_j^i.B[0]$ is the maximal value in the set $F_j^i.B$.

Thus, the running time of the GrA is

$$O\left(\sum k_j \min\{A, \max_{j=1,2,\ldots,n,\ i=1,2,\ldots,k_j} \{|F_j^i.B|\}\}\right).$$

It is easy to show that this running time can be rewritten as

$$O\left(\sum k_j \min\{A, \max_{j=1,2,\ldots,n} \{|F_j.B|\}\}\right).$$

□

In [9], we illustrate the idea of the GrA by means of a numerical example in more detail.

4 An FPTAS based on the GrA

In this section, a fully polynomial-time approximation scheme (FPTAS) is derived based on the GrA presented in Section 3.

Let $LB = \max_{j=1,\ldots,n} f_j(A)$ be a lower bound and $UB = n \cdot LB$ be an upper bound on the optimal objective function value.

The idea of the FPTAS is as follows. Let $\delta = \frac{\varepsilon LB}{n}$. To reduce the time complexity of the GrA, we have to diminish the number of columns $|F_j.B|$ considered, which corresponds to the number of different objective function values $b \in F_j.B, b \leq UB$. If we do not consider the original values $b \in F_j.B$ but the values $\overline{b}$ which are rounded up or down to the nearest multiple of δ values b, there are no more than $\frac{UB}{\delta} = \frac{n^2}{\varepsilon}$ different values $\overline{b}$. Then we will be able to approximate the function $F_j(t)$ into a similar function with no more than $2\frac{n^2}{\varepsilon}$ break points. Furthermore, for such a modified table representing a function $\overline{F}_j(t)$, we will have

$$|F_j(t) - \overline{F_j}(t)| < \delta \leq \frac{\varepsilon F(\pi^*)}{n}.$$

If we do the rounding and modification after each step [2.2.], the cumulative error will be no more than $n\delta \leq \varepsilon F(\pi^*)$, and the total running time of the n runs of the step [2.2.] will be

$$O\left(\frac{n^2 \sum k_j}{\varepsilon}\right),$$

i.e., an FPTAS is obtained.

In [7], a technique was proposed to improve the complexity of an approximation algorithm for optimization problems. This technique can be described as follows. If there exists an FPTAS for a problem with a running time bounded by a polynomial $P(L, \frac{1}{\varepsilon}, \frac{UB}{LB})$, where L is the length of the problem instance and UB, LB are known upper and lower bounds, and the value $\frac{UB}{LB}$ is not bounded by a constant, then the technique enables us to find in $P(L, \log\log\frac{UB}{LB})$ time values UB_0 and LB_0 such that

$$LB_0 \leq F^* \leq UB_0 < 3LB_0,$$

i.e., $\frac{UB_0}{LB_0}$ is bounded by the constant 3. By using such values UB_0 and LB_0, the running time of the FPTAS will be reduced to $P(L, \frac{1}{\varepsilon})$, where P is the same polynomial. So, by using this technique, we can improve the FPTAS to have a running time of

$$O\left(\frac{n \cdot \sum k_j}{\varepsilon}(1 + \log\log n)\right).$$

A detailed description of an FPTAS based on a GrA for some single machine scheduling problems was presented in [6]. In the following table these GrA and FPTAS are summarized.

Table 1: Time complexity of the GrA and FPTAS based on the GrA

Problem	Time complexity of the GrA	Time complexity of the FPTAS	Time complexity of the classical DPA
$1\|\|\sum w_j U_j$	$O(\min\{2^n, n \cdot \min\{d_{max}, F_{opt}\}\})$	-	$O(nd_{max})$
$1\|d_j = d'_j + A\|\sum U_j$	$O(n^2)$	-	$O(n\sum p_j)$
$1\|\|\sum GT_j$	$O(\min\{2^n, n \cdot \{d_{max}, nF^*\}\})$	$O(n^2 \log\log n + \frac{n^2}{\varepsilon})$	$O(nd_{max})$
$1\|\|\sum T_j$ special case $B-1$	$O(\min\{2^n, n \cdot \min\{d_{max}, F^*\}\})$	$O(n^2/\varepsilon)$	$O(nd_{max})$
$1\|\|\sum T_j$ special case $B-1G$	$O(\min\{n^2 \cdot \min\{d_{max}, F^*\}\})$	$O(n^3/\varepsilon)$	$O(n^2 d_{max})$
$1\|d_j = d\|\sum w_j T_j$	$O(\min\{n^2 \cdot \min\{d, F^*\}\})$	$O(n^3/\varepsilon)$	$O(n^2 d_{max})$
$1(no\text{-}idle)\|\|\max\sum w_j T_j$	$O(\min\{2^n, n \cdot \min\{d_{max}, F^*, \sum w_j\}\})$	$O(n^2 \log\log n + \frac{n^2}{\varepsilon})$	$O(nd_{max})$
$1(no\text{-}idle)\|\|\max\sum T_j$	$O(n^2)$	-	$O(nd_{max})$

5 Concluding Remarks

In this paper, we used a graphical approach to improve a known pseudo-polynomial algorithm for the Investment Optimization Problem and to derive a FPTAS with the best known running time.

The graphical approach can be applied to problems, where a pseudo-polynomial algorithm exists and Boolean variables are used in the sense that yes/no decisions have to be made. However, for the knapsack problem, the graphical algorithm mostly reduces substantially the number of states to be considered but the time complexity of the algorithm remains pseudo-polynomial. For example, for the single machine problem of maximizing total tardiness, the complexity of the graphical algorithm is $O(n^2)$, while the complexity of the dynamic programming algorithm is $O(n \sum p_j)$. Thus, the graphical approach is not only of a practical but also of a theoretical importance.

References

1. A.A. Lazarev and F. Werner, A Graphical Realization of the Dynamic Programming Method for Solving NP-hard Problems. Computers & Mathematics with Applications. Vol. 58, No. 4, 2009, 619 – 631.
2. A.V. Aho, J.E. Hopcroft and J.D. Ullman, Data Structures and Algorithms, Addison-Wesley, London, 1983.
3. H. Kellerer, U. Pferschy and D. Pisinger, Knapsack Problems, Springer-Verlag, Berlin, 2004.
4. D.X. Shaw and A. P. M. Wagelmans, An Algorithm for Single-Item Capacitated Economic Lot Sizing with Piecewise Linear Production Costs and General Holding Costs, Management Science, Vol. 44, No. 6, 1998, 831–838.
5. S. Kameshwaran and Y. Narahari, Nonconvex Piecewise Linear Knapsack Problems, European Journal of Operational Research, 192, 2009, 56–68.
6. E.R. Gafarov, A. Dolgui and F. Werner F.: Dynamic Programming Approach to Design FPTAS for Single Machine Scheduling Problems, Research Report LIMOS UMR CNRS 6158, 2012.
7. S. Chubanov, M.Y. Kovalyov and E. Pesch, An FPTAS for a Single-Item Capacitated Economic Lot-Sizing Problem with Monotone Cost Structure, Math. Program., Ser. A 106, 2006, 453 – 466.
8. K. Schmelev, X. Delorme, A. Dolgui, F. Grimaud and M.Y. Kovalev: Lot-Sizing on a Single Machine, ILP Models (submitted in May 2012).
9. E.R. Gafarov, A.A. Lazarev, A. Dolgui and F. Werner, An Improved Graphical Approach for an Investment Optimization Problem: Algorithm and a Numerical Example, Preprint 02/13, FMA, Otto-von-Guericke-Universität Magdeburg, 2013, 20 pages.

MISTA 2013

A Decomposition of the Max-min Fair Curriculum-based Course Timetabling Problem: The Impact of Solving Subproblems to Optimality

Moritz Mühlenthaler · **Rolf Wanka**

Abstract We propose a decomposition of the max-min fair curriculum-based course timetabling (MMF-CB-CTT) problem. The decomposition models the room assignment subproblem as a generalized lexicographic bottleneck optimization problem (GLBOP). We show that the GLBOP can be solved in polynomial time if the corresponding sum optimization problem can be solved in polynomial time as well. Thus, the room assignment subproblem of the MMF-CB-CTT problem can be solved efficiently. We apply this result to a previously proposed heuristic algorithm for the MMF-CB-CTT problem, in which solving the room assignment subproblem is a key ingredient. Our experimental results indicate that using the proposed decomposition improves the performance of the algorithm on most of the 21 ITC2007 test instances with respect to the quality of the best solution found and the average solution quality. Furthermore, we introduce a measure for the quality of a solution to a (generalized) lexicographic bottleneck optimization problem. This measure helps to overcome some limitations imposed by the qualitative nature of max-min fairness and aids the statistical evaluation of the performance of randomized algorithms for such problems.

1 Introduction

Many combinatorial problems can be solved effectively by decomposing them into subproblems which are tackled individually. Natural examples of successful decomposition approaches are divide-and-conquer algorithms for sorting and searching [19]. For many hard resource allocation and scheduling problems there is also some gain in dividing them into subproblems, even if the subproblems themselves are hard to solve or interact in some way. Basically, any approach that solves such a problem in two or more steps implicitly or explicitly decomposes it into subproblems. In the literature, there are numerous examples of multi-step and other decomposition approaches to solving such combinatorial problems; applications include timetabling [7,21], crew and staff scheduling [35,11], and production scheduling [14,16].

Research funded in parts by the School of Engineering of the University of Erlangen-Nuremberg.

Moritz Mühlenthaler · Rolf Wanka
University of Erlangen-Nuremberg, Department of Computer Science
E-mail: {moritz.muehlenthaler | rolf.wanka}@cs.fau.de

We consider a decomposition approach to a variant of the curriculum-based course timetabling (CB-CTT) problem. The CB-CTT problem has been proposed in the context of the timetabling competition ITC2007 [25] and has since then received a great deal of attention in the timetabling community. The CB-CTT problem models the task of assigning timeslots and rooms to courses in the setting of a university and includes a number of requirements that typically arise in real-world course timetabling applications. For instance, courses occurring in the same curriculum must not be taught at the same time and courses should be assigned to rooms with a sufficient number of seats. The combinatorial structure of the problem is quite complex and has been investigated, for example, using polyhedral studies [8,21]. Due to the structural complexity, solving instances with a large number of courses and curricula is typically out of reach for exact methods. However, a wealth of (meta-)heuristic methods have been applied successfully to generate high quality timetables, even for large CB-CTT instances, see for example [1,23,27]. In this work, our focus is the max-min fair curriculum-based course timetabling (MMF-CB-CTT) problem introduced in [26], which replaces the original optimization objective by a lexicographic bottleneck objective. The goal of the MMF-CB-CTT problem formulation is to favor *fair* timetables and thus improve the overall stakeholder satisfaction. The underlying fairness concept is (lexicographic) max-min fairness.

It is a common technique to decompose the CB-CTT problem into subproblems which are easier to handle individually [21,23]. The usual approach is to perform room and timeslot assignments separately, but other approaches have been explored as well [7]. Applying the usual decomposition approach yields as subproblems a bounded list coloring problem for the timeslot assignment and, for each timeslot, a linear sum assignment problem (LSAP) for assigning courses to rooms [8,21]. Unfortunately, there are dependencies between LSAPs for different timeslots, so an optimal room assignment can only be obtained for a single timeslot, while the rest of the timetable remains fixed. We show that for an analogous decomposition of the MMF-CB-CTT problem, the room assignment subproblem for a single timeslot can also be solved efficiently by modeling it as a generalized lexicographic bottleneck assignment problem. In a more general setting, we show that generalized lexicographic bottleneck optimization problems can be solved efficiently if the corresponding sum optimization problem can be solved efficiently.

Furthermore, we propose a new measure for the quality of a solution to an optimization problem with a max-min fairness/lexicographic bottleneck objective such as the MMF-CB-CTT problem. This measure helps to overcome some limitations imposed by the qualitative nature of max-min fairness.

We evaluate the quality of the timetables produced by the algorithm MAXMINFAIR_SA from [26], with and without the new decomposition. We use our aforementioned new quality measure to compute the average solution quality. In the original algorithm the room assignment subproblem was modeled as an LSAP. However, an optimal solution to the LSAP is not necessarily optimal for the room assignment subproblem of the MMF-CB-CTT problem. Our experiments indicate that making use of the new decomposition improves the best timetables found on 18 out of 21 CB-CTT instances from the ITC2007 competition. The new decomposition results in an improved average solution quality for 16 out of 21 instances. According to the Wilcoxon rank-sum test (one-sided, significance level 0.01) MAXMINFAIR_SA with the new decomposition is significantly better than the original approach on 12 instances.

The remainder of this work is organized as follows: In Section 2, we provide relevant background on the CB-CTT and MMF-CB-CTT problem formulations, as well as max-min fairness and the assignment problem. In Section 3, we introduce the decomposition of the

MMF-CB-CTT problem. We propose the measure for the amount of max-min fairness of an allocation in Section 4. Section 5 presents our evaluation of the performance impact of the decomposition for MMF-CB-CTT problems. Section 6 contains concluding remarks.

2 Background

In this section we will provide some relevant background on fair resource allocation, university course timetabling and the assignment problem.

2.1 Fair Resource Allocation

Fairness comes into play when scarce resources are distributed over a set of stakeholders with demands. The topic of fairness and how to measure it has received great attention for example in the area of economics, where the distribution of wealth and income is of interest [12]. In computer science, fairness aspects have been studied for example in the design of network communication protocols, in particular in the context of bandwidth allocation and traffic shaping [22, 15]. Fairness aspects have been addressed explicitly for example for various kinds of scheduling problems, including personnel scheduling [33], sports scheduling [31], course scheduling [26] and aircraft scheduling [34]. In the context of resource allocation allocation in general, fairness has been studied in [3, 28]. Approximation algorithms for fair optimization problems have been studied in [18, 20].

In the next sections of this work we will deal with a fair variant of a university course timetabling problem from [26] that builds on the notion of max-min fairness. A resource allocation is called *max-min fair* if an improvement for any stakeholder is possible only at the expense of another stakeholder who is worse-off. Consider the problem of allocating resources to n stakeholders. A resource allocation induces an allocation vector $X = (x_1, \ldots, x_n)$, where x_i, $1 \leq i \leq n$, corresponds to the amount of resources allocated to stakeholder i. Let $\vec{X}$ denote the sequence containing the entries of an allocation vector X sorted in non-decreasing order. Likewise, let $\overleftarrow{X}$ denote the sequence containing the entries of X sorted in non-increasing order. For allocation vectors X and Y induced by two solutions to a minimization (maximization) problem, we write $X \preceq Y$ ($X \preceq_{max} Y$) if X is at least as fair as Y with respect to max-min fairness. That is, $X \preceq Y$ holds iff $\vec{X} \preceq_{lex} \vec{Y}$, and $X \preceq_{max} Y$ holds iff $\vec{Y} \preceq_{lex} \vec{X}$. A solution to a resource allocation problem is called *max-min fair* if its induced allocation vector is at least as fair as the induced allocation vector of any other feasible solution.

Max-min fairness enforces an efficient resource usage to some extent, since an improved resource utilization is accepted to the benefit of a stakeholder as long as it is not at the expense of another stakeholder who is worse-off. Hence, a max-min fair solution to a resource allocation problem is Pareto-optimal. One of the limitations of max-min fairness is that the concept is purely qualitative, i. e., given two allocation vectors X and Y, max-min fairness just determines which of the two is fairer, but not by how much. In order to aid the statistical evaluation of the performance of algorithms for max-min fair resource allocation problems, in Section 4 we will introduce a metric for the difference in quality between two allocation vectors which is compatible with the $\preceq$-relation.

2.2 Curriculum-based Course Timetabling

The academic course timetabling problem captures the task of assigning a set of courses to rooms and timeslots in the setting of a university. In Section 3 we will focus on decompositions of two particular variants of the academic course timetabling problem: the curriculum-based course timetabling (CB-CTT) problem from track three of the second international timetabling competition [10], and its max-min fair version, MMF-CB-CTT, proposed in [26]. The CB-CTT formulation has attracted a great deal of interest in the research community as well as in practical course timetabling [23,21,32]. The max-min fair variant differs from the basic CB-CTT formulation only with respect to the objective function. We will now introduce some terminology and state definitions relevant to the later sections of this work.

A CB-CTT instance consists of a set of courses, a set of curricula, a set of rooms, a set of teachers and a set of days. Each day is divided into a fixed number of timeslots; a day together with a timeslot is referred to as a *period*. A period in conjunction with a room is called a *resource*. Each course consists of a set of events that need to be scheduled, is taught by a teacher and has a fixed number of students attending it. A course can only be taught in certain available periods. Each curriculum is a set of courses, no two of which may be taught in the same period. Each room has a *capacity*, a maximum number of students it can accommodate. A solution to a CB-CTT instance is a *timetable*, i. e., an assignment of the courses to the resources subject to a number of hard and soft constraints. A timetable that satisfies all hard constraints is *feasible*.

Later on, we will deal exclusively with feasible timetables, so we will not cover the evaluation of hard constraints at all (please refer to [10] a detailed description). However, some understanding of the soft constraint evaluation will be useful later on, so we will touch on this very briefly. The CB-CTT problem formulation features the following soft constraints:

S1 *RoomCapacity*: Each lecture should be assigned to a room of sufficient size.
S2 *MinWorkingDays*: The individual lectures of each course should be distributed over a certain minimum number of days.
S3 *IsolatedLectures*: For each curriculum, all courses in the curriculum should be scheduled in adjacent periods.
S4 *RoomStability*: The lectures of each course should be held in the same room.

The violation of a soft constraint results in a "penalty" for the timetable. The total penalty of a timetable τ is aggregated by the objective function c which just sums the penalties for the individual soft constraint violations:

$$c(\tau) = \sum_{1 \le i \le 4} c_{Si}(\tau), \tag{1}$$

where $c_{S1}, \ldots, c_{S4}$ are the penalties determined by the soft constraints (S1)–(S4). The relative importance of the different soft constraints is set by a weight factor for each soft constraint. Since the weights will be of no relevance to our arguments later on, we assume that appropriate weighting has been applied within $c_{S1}, \ldots, c_{S4}$. A detailed specification of the penalty functions can be found in [10].

Definition 1 (Curriculum-based Course Timetabling Problem) Given a CB-CTT instance I, find a feasible timetable τ such that $c(\tau)$ is minimal.

A max-min fair variant of the CB-CTT problem was defined in [26]. Given a CB-CTT instance with curricula $u_1, \ldots, u_k$. The allocation vector of a timetable τ is given by:

$$A(\tau) = (c(u_1, \tau), c(u_2, \tau), \ldots, c(u_k, \tau)) \ , \tag{2}$$

where $c(u_j, \tau) = \sum_{1 \leq i \leq 4} c_{Si}(u_j, \tau)$, $i \in \{1, \ldots, 4\}$, is the CB-CTT objective function restricted to the events of the courses in curriculum u_j, $j \in \{1, \ldots, k\}$.

Definition 2 (Max-min Fair Curriculum-based Course Timetabling Problem) Given a CB-CTT instance I, find a feasible timetable τ such that $A(\tau)$ is max-min fair.

2.3 The Assignment Problem

The assignment problem is a classical problem in combinatorial optimization which appears in many applications, for example personnel scheduling, job scheduling and object tracking, just to name a few. For a comprehensive overview of the body of research on the assignment problem and the applications see [5,30]. In CB-CTT problem for example, the assignment problem appears as a subproblem [21,24]. There exist polynomial-time algorithms for many variants of the assignment problem.

Let A and B be two sets of cardinality n. An assignment of the elements of A to the elements of B is a bijection $\sigma : A \to B$. Typically, assignment problems are optimization problems, i. e., among all bijections from A to B, we are looking for one that is optimal with respect to a certain objective function. In the context of (fair) curriculum-based course timetabling, we are in particular interested in two variants of the assignment problem, namely the linear assignment problem (LSAP) and the lexicographic bottleneck assignment problem (LBAP).

Definition 3 (Linear Sum Assignment Problem (LSAP)) Given a cost function $c : A \times B \to \mathbb{R}$, find a bijection $\sigma : A \to B$ such that $\sum_{a \in A} c(a, \sigma(a))$ is minimal.

There exist various algorithms for solving LSAPs efficiently, including the well-known Hungarian algorithm [29, p. 248ff] and network flow algorithms [13]. In the following, let $T_{\mathrm{LSAP}}(n)$ be the time complexity of solving an LSAP instance with $|A| = |B| = n$.

When solving MMF-CB-CTT problems using the decomposition proposed in the next section, the task of finding max-min fair assignments occurs as a subproblem. An assignment $\sigma : A \to B$ is called max-min fair, if for any assignment $\sigma' : A \to B$ we have $\mathbf{c}(\sigma) \preceq \mathbf{c}(\sigma')$, where $\mathbf{c}(\sigma) = (c(a, \sigma(a)))_{a \in A}$ and $\preceq$ is the max-min fair comparison from Section 2.1.

Definition 4 (Lexicographic Bottleneck Assignment Problem (LBAP)) Given a cost function $c : A \times B \to \mathbb{R}$, find a max-min fair bijection $\sigma : A \to B$.

A LBAP can be transformed into a LSAP by scaling the cost values appropriately. This results in an exponential blow-up of the cost values, which may be undesirable in practical applications [4]. Alternatively, an LBAP can be reduced to a *linear vector assignment problem*, which belongs to the class of *algebraic assignment problems* [6]. Using this reduction, a given LBAP with a cost function c can be solved in time $O(kn^3)$, where k is the number of distinct values attained by c [9]. The reduction is straightforward: For each $i \in \{1, \ldots, k\}$, let $\mathbf{e}_i \in \mathbb{N}^k$, be a vector whose i-th component is 1 and all other components are 0. The cost function c is replaced by a vector-valued function $c' : A \times B \to \mathbb{N}^k$ such that $c'(a, \sigma(a)) = \mathbf{e}_i$ if $c(a, \sigma(a))$ is the i-th largest value attained by c. An assignment σ that yields a lexicographically minimal cost vector $\sum_{a \in A} c'(a, \sigma(a))$ is an optimal solution to the corresponding LBAP.

3 Problem Decomposition

It is a common approach to decompose the CB-CTT problem in a way that room and timeslot assignment is preformed separately, see for example [21,24]. For courses in a single timeslot, an optimal room assignment can be determined by solving a LSAP instance. In this section, we establish a similar result for the MMF-CB-CTT problem: An optimal room assignment for the courses in a single timeslot can be determined by solving a (generalized) LBAP instance. In the spirit of Benders decomposition approach [2], we further consider decompositions of sum optimization problems into master and subproblems such that the subproblems can be solved efficiently. We derive sufficient conditions under which such a decomposition of a sum optimization problem carries over to its lexicographic bottleneck counterpart.

Let the sum optimization problem (SOP) be the following combinatorial problem:

$$\begin{aligned} \text{(SOP)} \quad & \min c \cdot x \\ & \text{s.t. } x \in X \ , \end{aligned}$$

where $c \in \mathbb{N}^m$ and $X \subseteq \{0,1\}^m$ and X contains all feasible assignments of the variables $x_1, \ldots, x_m$. For the LSAP from Section 2.3 for example, X encodes the bijections $A \to B$. Let T_X be the time required to solve the SOP on the combinatorial structure X. Each elementary operation (e.g., addition) in the solution procedure is assumed to take constant time. Now consider the following generalized lexicographic bottleneck optimization problem (GLBOP):

$$\begin{aligned} \text{(GLBOP)} \quad & \min_{\preceq} f(x) = (w_1 x_{i_1}, \ldots, w_\ell x_{i_\ell}) \\ & \text{s.t. } x \in X \ , \end{aligned} \tag{3}$$

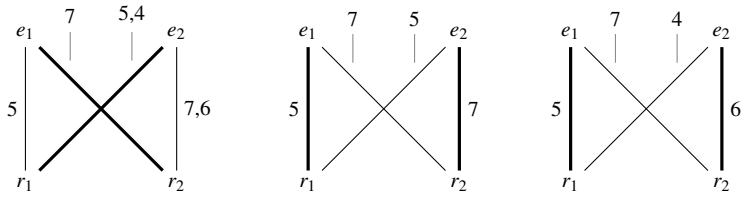
where $f : X \to \mathbb{N}^\ell$ for some $\ell \in \mathbb{N}$, $w_1, \ldots, w_\ell \in \mathbb{N}$ are called weights and $i_1, \ldots, i_\ell \in \{1, \ldots, m\}$. The minimum is taken with respect to the relation $\preceq$ from Section 2.1. Please note that this is a generalization of the definition of the LBOP in [4,9] since each of the variables can appear in several components of f.

Theorem 1 *A GLBOP instance with ℓ weights, t of them distinct, can be solved in time $O(\ell^2 + t \cdot T_X)$.*

Proof Following the vectorial approach of Della Croce et al. in [9], we reduce the GLBOP to a vectorial sum optimization problem (VSOP). We construct a cost matrix $Z \in \mathbb{N}^{m \times t}$, which contains in each row i a cost vector for the variable x_i. With each weight w we associate a cost vector $(d_1, \ldots, d_t) \in \mathbb{N}^t$ such that $d_i = 1$, if w is the i-th largest weight, and $d_i = 0$ otherwise. Let the i-th row of the cost matrix Z be the sum of all cost vectors associated to the weights that occur in a product with x_i in any component of f. The VSOP has the following form:

$$\begin{aligned} \min_{\preceq_{lex}} \quad & \tilde{f}(x) = x \cdot Z \\ & \text{s.t. } x \in X \ . \end{aligned} \tag{4}$$

Suppose for a contradiction that an optimal solution x^* to the VSOP (4) is not optimal for the GLBOP (3). Then there is a solution $x' \in X$ such that $f(x') \prec f(x^*)$. As a consequence, $\tilde{f}(x') \prec_{lex} \tilde{f}(x^*)$ due to the construction of the cost vectors above. This is a contradiction to



(a) Room assignment problem: optimal cost 7, 5, 4

(b) LBAP 1: optimal cost 7,5

(c) LBAP 2: optimal cost 6,5

Fig. 1: A room assignment problem examples with two courses, e_1 and e_2, and two rooms r_1 and r_2. None of the two LBAPs (b) and (c) models the room assignment problem (a): The optimal solutions highlighted for LBAPs are not optimal for the room assignment problem.

the optimality of x^*. The VSOP (4) can be solved in time $O(t \cdot T_X)$ because each elementary operation involving the weights is now performed on a vector of length t. The time complexity of the reduction is $O(\ell^2)$.

A similar result can be obtained by modifying the cost scaling approach in [4]: Taking the sum of the scaled weights that occur in a product involving the variable x_i is equivalent to adding the corresponding cost vectors during the construction of the cost matrix Z in the proof of Theorem 1. This approach results in a direct reduction of the GLBOP to the SOP with weights that are exponentially large compared to the original weights.

We will now apply the result above to a decomposition of the MMF-CB-CTT problem. The decomposition isolates, for a single period, the assignment of courses to rooms, from the rest of the problem. So, the task is to find an optimal room assignment for a given period assuming the rest of the timetable is fixed. Optimizing the rest of the timetable can be considered the master problem. For the CB-CTT problem the room assignment subproblem is a LSAP and hence can be solved efficiently. Our goal is to show that the room assignment subproblem of the MMF-CB-CTT problem can be solved efficiently as well. In the following, let C be the set of courses, R be the set of rooms, P be the set of periods, and $U \subseteq \mathscr{P}(C)$ be the curricula. By C_p we denote the set of courses scheduled in the period $p \in P$. Please note that C_p is determined by the solution of the master problem. Furthermore, let $U_e = \{u \in U \mid e \in u\}$.

Figure 1 illustrates why simply solving a LBAP does not account for the intricacies of the room assignment subproblem of the MMF-CB-CTT problem. To keep things simple, there is only a single period p, two courses $C = C_p = \{e_1, e_2\}$, and two rooms $R = \{r_1, r_2\}$. However, the course e_2 is in two curricula and thus determines two entries of the overall allocation vector. The cost on each edge connected to e_1 shows the costs generated for each of the two curricula by assigning a particular room to e_2. Figures 1b and 1c are LBAP instances that reflect only costs for one of the two curricula of e_2. The assignments highlighted are both optimal solutions to the individual LBAPs, but none of them is optimal for the room assignment shown in Figure 1a.

Theorem 2 *The room assignment subproblem of the MMF-CB-CTT problem is a GLBOP.*

Proof The room assignment subproblem of the CB-CTT problem, for a fixed period p, is a LSAP. Therefore, restricting the set of courses to a single curriculum $u \in U$, also yields an LSAP as room assignment subproblem, however, with at most one course of u in C_p

(due to the conflict constraints). Thus, assigning a course e to a room r in p completes the timetable τ from the perspective of all curricula in U_e and therefore determines their cost entries $w_{u,e,r} = c(u,\tau)$ for each $u \in U_e$ in the allocation vector (2). Thus, the room assignment subproblem can be written as

$$\min_{\preceq} \quad \prod_{e,r} (w_{u,e,r} \cdot x_{e,r})_{u \in U_e} \tag{5}$$
$$\text{s. t. } x \in X \quad ,$$

where $\prod$ denotes the concatenation of tuples and X contains all valid room assignments of the courses in period p to the rooms R. The variables are $\{x_{e,r}\}_{e \in C_p, r \in R}$ and $x_{e,r} = 1$ if course e is assigned to room r in period p, and 0 otherwise. X contains all feasible variable assignments. Clearly, problem (5) is a GLBOP.

Corollary 1 *For a period p, the room assignment subproblem* (5) *with* $n = |C_p|$ *courses can be solved in time* $O(|U|^2 + |U| \cdot T_{\text{LSAP}}(n))$.

Proof Problem (5) is an assignment problem, just as the room assignment subproblem of the CB-CTT problem. Only the objective function is different. Hence, combining Theorems 1 and 2 yields the result.

Remark 1 As noted by Lach and Lübbecke in [21], the room stability constraint (S4) introduces dependencies between room assignments in different timeslots, which prevents us from extending the decomposition to more than a single period.

Remark 2 In the general case, whenever there is a decomposition of a SOP into a master problem and a subproblem which can be solved efficiently, the subproblem of the corresponding LBOP can be solved efficiently if it is a GLBOP. Since complex SOPs are much better studied than LBOPs this observation may be useful when adding max-min fairness objectives to existing SOPs.

In Section 5 we will provide experimental evidence that solving the room assignment subproblem to optimality is useful for improving the performance of a heuristic algorithm for the MMF-CB-CTT problem.

4 Quantifying Max-min Fairness

When dealing with randomized optimization algorithms, one can employ a wealth of statistical tools to extract meaningful information about algorithms' absolute and relative performance. These tools include statistical tests such as the Wilcoxon rank-sum test and measures such as the mean quality of the solutions, the standard deviation, the median quality, the quality of the best and worst solutions, and so on. Due to the qualitative nature of max-min fairness, so far only statistical tools based on ranking can be used for evaluating randomized max-min fair optimization algorithms. In this section we propose a novel approach to partially overcome this limitation. We consider max-min fair minimization problems whose solutions induce allocation vectors in $\mathbb{N}_0^n$. The main idea is to construct an isomorphism from the set of sorted allocation vectors to an interval in the natural numbers ordered by the usual $\leq$ relation. Using this isomorphism, we can perform all operations on natural/real numbers, round the result to the nearest integer, and retrieve the corresponding allocation vector. This means that if we have a set of allocation vectors, we can determine for example an allocation

vector close to the average allocation. Thus, in our experiments in the next section we will be able to compare the average solution quality of two algorithms for the MMF-CB-CTT problems.

Lemma 1 *Let* $\mathbb{A}_n = \{(x_1,\ldots,x_n) \in \mathbb{N}_0^n \mid x_1 \geq x_2 \geq \ldots \geq x_n\}$. *Let the function* $\text{rank} : \mathbb{A}_n \to \mathbb{N}_0$ *be*

$$\text{rank}(x_1,\ldots,x_n) = \sum_{i=1}^{n} \binom{n+x_i-i}{x_i-1} \ , \tag{6}$$

where $\binom{k}{-1} = 0$ *for all* $k \in \mathbb{N}_0$. *Then* $\text{rank} : (\mathbb{A}_n, \preceq_{lex}) \to (\mathbb{N}_0, \leq)$ *is an isomorphism.*

Proof $(\mathbb{A}_n, \preceq_{lex})$ is a linearly ordered set with least element $(0,\ldots,0)$. Therefore, for each $X \in \mathbb{A}_n$ there is a unique natural number $r_X = |\{Y \in \mathbb{A}_n \mid Y \prec_{lex} X\}|$. Thus, the function mapping each $X \in \mathbb{A}_n$ to r_X is a bijective mapping and it is order-preserving as required. It remains to be shown that $\text{rank}(X)$ computes r_X for all $X \in \mathbb{A}_n$.

Let $X = (x_1,\ldots,x_n) \in \mathbb{A}_n$. The value r_X can be determined by the following recursion:

$$r_{x_1,\ldots,x_n} = r_{x_2,\ldots,x_n} + r_{x_1,0,\ldots,0} \ . \tag{7}$$

This recursion separately counts the non-increasing sequences $\{Y \in \mathbb{A}_n \mid (x_1,0,\ldots,0) \preceq_{lex} Y \prec_{lex} X\}$ and $\{Y \in \mathbb{A}_n \mid Y \prec_{lex} (x_1,0,\ldots,0)\}$. The number of sorted sequences of length n over an ordered alphabet of size k is $\binom{n+k}{k}$. Thus,

$$r_{\underbrace{x_1,0,\ldots,0}_{n}} = \binom{n+x_1-1}{x_1-1} \ .$$

In particular, for $x \in \mathbb{N}_0$ we have $r_x = \binom{x}{x-1} = x$. Unfolding the recursion (7) yields (6). Therefore, $\text{rank}(X)$ computes r_X.

The relation $\preceq$ gives rise to an equivalence relation $\sim$ on $\mathbb{N}_0^n$, the set of allocation vectors of dimension n: Two allocation vectors X and Y are equivalent if $X \preceq Y$ and $Y \preceq X$. We denote by $[X]_\sim$ the equivalence class containing X.

Theorem 3 *Let* $X \in \mathbb{N}_0^n$. *Then the function*

$$\begin{aligned} \rho : (\mathbb{N}_0{}^n_{/\sim}, \preceq) &\to (\mathbb{N}_0, \leq) \\ [X]_\sim &\mapsto \text{rank}(\overleftarrow{X}) \ , \end{aligned}$$

is an isomorphism.

Proof For each equivalence class, we can choose as a representative the unique allocation whose elements occur in non-increasing order, and use the isomorphism rank from Lemma 1.

We will now consider maximization problems and the corresponding max-min fair comparison $\preceq_{max}$. In a similar fashion to $\preceq$, it gives rise to an equivalence relation $\sim_{max}$ on $\mathbb{N}_0^n$. We will show that, if there is a certain maximum amount resources for all stakeholders, then we can use the function rank again to construct an isomorphism between allocation vectors and a subset of the natural numbers. The assumption of a maximum amount of resources is necessary, since, if the resource allocated to a stakeholder are unbounded, the number of fairer and unfairer allocations are infinite. The isomorphism is useful in order to measure the quality and the quality difference of allocation vectors for maximization problems.

Theorem 4 *Let $m,n \in \mathbb{N}_0$, $D = \{0,\ldots,m\}^n$, $M = (m,\ldots,m) \in D$, and $I = \{\text{rank}(M - \vec{X}) \mid X \in D\}$. Then the function*

$$\begin{aligned} \rho' : (D_{/\sim_{max}}, \preceq_{max}) &\to (I, \leq) \\ [X]_{\sim_{max}} &\mapsto \text{rank}(M - \vec{X}) \ , \end{aligned}$$

is an isomorphism. □

5 Evaluation

In this section we are going to present experimental evidence for the usefulness decomposition presented in Section 3. We compare the performance of two randomized heuristic algorithms for the MMF-CB-CTT problem, both of which are based on the algorithm MAXMINFAIR_SA from [26]. The first algorithm is the one that performed best in [26]. It uses a decomposition of the MMF-CB-CTT problem that is similar to the one presented in this work, but models the room assignment subproblem as LSAP. Thus, we will refer to this algorithm by MMF_SA_LSAP. The second algorithm, MMF_SA_GLBOP, uses the MMF-CB-CTT decomposition from Section 3 and thus solves a GLBOP to obtain an optimal room assignment for a given period. Apart from how the room assignment subproblem is solved, the the two algorithms are identical. We compare both algorithms with respect to the best solutions they produce as well as the average solution quality per instance. In order to determine average allocations, we use the isomorphism ρ from Theorem 3 as described in the previous section. Our results indicate that the algorithm which makes GLBOP room assignment significantly outperforms the other one.

The algorithm MAXMINFAIR_SA is a variant of simulated annealing (SA) [17] which has been tailored to the MMF-CB-CTT problem. Simulated annealing iteratively generates new candidate solutions and keeps (or *accepts*) a new solution if it is better. If the new solution is worse, then it is accepted with a certain probability which depends on the temperature ϑ. There are three crucial design choices when adapting simulated annealing to a particular problem: The cooling schedule, the acceptance criterion, and the neighborhood structure. In both algorithms under consideration, MMF_SA_GLBOP and MMF_SA_LSAP, we use the standard geometric cooling schedule, which lets the temperature decay exponentially from a given ϑ_{max} to a given ϑ_{min}. Both algorithms use the acceptance criterion based on the component-wise energy difference, which performed best in a comparison of different acceptance criteria in [26]. Also, both algorithms use a neighborhood structure based on the well-known Kempe-move. A Kempe-move swaps a subset of the events assigned to two given periods such that the conflict constraints between the events are not violated. The difference between both algorithms is how the room-assignment subproblems are solved after performing a Kempe-move: MMF_SA_LSAP solves two LSAPs in order to assign rooms efficiently, while MMF_SA_GLBOP solves two GLBOPs. Thus, the second room assignment performed by MMF_SA_GLBOP is optimal with respect to the rest of the timetable. Our evaluation shows that this is beneficial for the overall algorithm performance.

In order to compare the two algorithms, we performed 50 independent runs for each algorithm on the 21 CB-CTT instances from track three of the timetabling competition ITC2007 [10]. In each run, 1000000 iterations of the simulated annealing procedure were performed. We did not tweak the temperature-related parameters of the algorithms extensively, but determined experimentally that $\vartheta_{max} = 5$ and $\vartheta_{min} = 0.01$, as suggested in [26] work well. Table 1 shows the best allocation vectors obtained by MMF_SA_LSAP and

MMF_SA_GLBOP on the 21 CB-CTT instances. To aid the presentation of the results, Table 1 shows the allocation vectors in a compressed form: The penalty values are sorted in non-increasing orders and repetitions of values are shown as exponents. For example, a table entry of $5^2 10^1 1$ denotes an allocation vector in which the penalty value 5 appears two times, 1 appears one and 0 eleven times. Note that from the max-min fairness perspective it is not important to which curriculum a penalty value corresponds, so we omit this information. The results in Table 1 show that MMF_SA_GLBOP finds better solutions than MMF_SA_LSAP on 18 instances, while the best solutions found by MMF_SA_LSAP are better on `comp04` and `comp18`. In addition to the best allocation vectors, Table 1 shows the average allocations over the 50 runs for each instance. The average allocations have been computed using ρ from Section 4: The allocation vectors were mapped to the natural numbers, then the average was calculated and rounded to the nearest integer. Finally, the result was mapped back to the corresponding equivalence class of allocation vectors. A comparison of the average allocations shows that in this respect, MMF_SA_GLBOP outperforms MMF_SA_LSAP on 16 instances while it is beaten on the instances `comp05`, `comp08`, `comp15` and `comp21`.

We also performed the one-sided Wilcoxon rank-sum test with a significance level of 0.01. According to the test, MMF_SA_GLBOP yields significantly better results than MMF_SA_LSAP on instances `comp02`, `comp06`, `comp07`, `comp08`, `comp10`, `comp13`, `comp14`, `comp17`, `comp18`, `comp19`, `comp20` and `comp21`. In contrast, MMF_SA_LSAP is not significantly better than MMF_SA_GLBOP on any of the 21 instances with this significance level. The results of the Wilcoxon test are consistent with the data in Table 1.

In contrast to experimental setup in [26], we did not use a timeout, but set a fixed number of iterations for the direct comparison of MMF_SA_GLBOP and MMF_SA_LSAP. The reason for this decision is that in practice, solving GLBOP instances takes significantly more time than solving LSAPs. The increase in runtime is due to overhead required by construction of the cost matrix ($O(|U|^2)$) and the $O(|U|)$ factor for solving the linear vector assignment problem (see Corollary 1). Since we are mainly interested in the implications of modelling the room assignment subproblem with a GLBOP instead of an LSAP, both algorithms should be able to solve a similar number of room-assignment subproblems. From the data shown in Table 1 we can conclude, that using the decomposition presented in Section 3 is clearly the smarter choice, since, after solving equally many subproblems, it produces superior results compared to the approach from [26]. However, if we employ the timeout as it was required for the ITC2007 competition (see [10]), MMF_SA_LSAP is the better choice, because it can perform significantly more iterations within the given timeout.

6 Conclusion

In this work we proposed a decomposition of the MMF-CB-CT problem from [26]. The decomposition models the room assignment subproblem as a assignment problem with a generalized lexicographic bottleneck objective. The decomposition enables us to compute an optimal room assignment for a given period in polynomial time. We use this result to improve the performance of the MaxMinFair_SA algorithm proposed in [26], which originally modelled the room assignment subproblem as an LSAP. An optimal solution to the LSAP however is not necessarily optimal for the room assignment subproblem of the MMF-CB-CTT problem. In our experiments we compare the performance of two variants of the MaxMinFair_SA algorithm, wich differ only with respect to how the room assignment subproblem is tackled. Our results indicate that solving the room assignment subproblem

Table 1: Comparison of the best and average allocation vectors of the solutions found by MMF_SA_LSAP and MMF_SA_GLBOP on the 21 CB-CTT instances from [25] for 50 independent runs per instance and per algorithm. For each instance, the best results and best average results are marked in bold face.

	MMF_SA_GLBOP		MMF_SA_LSAP	
Instance	best	average	best	average
comp01.ectt	$\mathbf{5^2,0^{12}}$	$\mathbf{6,5,4^5,3^4,2,1,0}$	$5^2,1,0^{11}$	$6,5^3,4,2^2,1^5,0^2$
comp02.ectt	$\mathbf{2^{32},1^5,0^{33}}$	$\mathbf{4^5,3^{18},2^{16},1^{18},0^{13}}$	$4,3^3,2^{26},1^{12},0^{28}$	$5^2,4^{30},3^4,2^{17},1^{16},0$
comp03.ectt	$\mathbf{6^4,4^{11},2^{23},1,0^{29}}$	$\mathbf{6^5,5^{40},4^8,3^2,2^4,0^9}$	$6^4,4^{12},2^{27},1^3,0^{22}$	$6^{10},5^9,4,3^{24},2,1^3,0^{20}$
comp04.ectt	$6^4,4^2,2^4,1^7,0^{40}$	$\mathbf{6^4,4^3,3^6,2^{34},1^8,0^2}$	$\mathbf{6^4,4^2,2^4,0^{47}}$	$6^4,4^3,3^{11},2^{33},1^4,0^2$
comp05.ectt	$\mathbf{19^2,18^3,17^3,16^5,15^2,14^8},\ldots$	$20^3,18^6,16,15^9,14^2,13^3,\ldots$	$19^2,18^3,17^3,16^6,15^2,14^7,\ldots$	$\mathbf{20^2,19^4,18^{11},16^3,15^{30},14^7},\ldots$
comp06.ectt	$\mathbf{12,4,2^{29},1^{15},0^{24}}$	$\mathbf{12,5^6,4^2,3^9,2^{43},1^4,0^5}$	$12,4^3,3,2^{31},1^{12},0^{22}$	$12,5^{11},4^3,3^5,2^4,1^5,0^{41}$
comp07.ectt	$\mathbf{6,2^{14},1^{18},0^{44}}$	$\mathbf{6,4,3^{21},2^9,1^{15},0^{30}}$	$6,3,2^{30},1^{25},0^{20}$	$6,4^2,3^{22},2^7,1^{11},0^{34}$
comp08.ectt	$\mathbf{6^4,4^2,2^9,1^6,0^{40}}$	$6^4,4^3,3^{23},2^{15},1^{14},0^2$	$6^4,4^2,2^{11},1^4,0^{40}$	$\mathbf{6^4,4^4,3^3,2^9,1^{20},0^{21}}$
comp09.ectt	$\mathbf{6^9,4^{11},2^{18},1^2,0^{35}}$	$\mathbf{6^{10},5^3,4^8,3^{21},2^{12},1^{13},0^8}$	$6^9,4^{14},2^{12},1^6,0^{34}$	$6^{10},5^5,4^4,3^{10},2^{10},1^{17},0^{19}$
comp10.ectt	$\mathbf{2^{12},1^5,0^{50}}$	$\mathbf{4,3^{34},2^{13},0^{19}}$	$2^{20},1^9,0^{38}$	$4^{10},3^4,2^{24},1^6,0^{23}$
comp11.ectt	0^{13}	0^{13}	0^{13}	0^{13}
comp12.ectt	$\mathbf{10^4,8^{26},7,6^{50},5^2,4^{38}},\ldots$	$\mathbf{12^2,10^2,9^{45},8^2,7^{12},6^{11}},\ldots$	$10^5,8^{26},6^{48},5^6,4^{42},3^4,\ldots$	$12^3,11^{19},10^9,9^2,8^{21},7^{12},\ldots$
comp13.ectt	$\mathbf{6^6,4^4,2^{12},1^2,0^{42}}$	$\mathbf{6^6,4^6,3^{11},2^{24},1^6,0^{13}}$	$6^6,4^4,2^{21},1^4,0^{31}$	$6^6,4^7,3^{11},2,1^{29},0^{12}$
comp14.ectt	$\mathbf{8^4,4^3,2^{13},1^3,0^{37}}$	$\mathbf{8^4,4^5,3^5,2^{32},1^{11},0^3}$	$8^4,4^3,2^{16},1^2,0^{35}$	$8^4,4^{13},3^{22},2^{10},1^7,0^4$
comp15.ectt	$\mathbf{6^4,4^{10},2^{22},1^4,0^{28}}$	$6^{11},5^3,4^8,3^{13},2^7,1^2,0^{24}$	$6^4,4^{11},3,2^{25},1^5,0^{22}$	$\mathbf{6^8,5^3,4^{18},3,2^{12},1^{18},0^8}$
comp16.ectt	$\mathbf{4^5,2^{14},1^3,0^{49}}$	$\mathbf{5^5,4^{10},3^6,2^{19},1^6,0^{25}}$	$4^5,3,2^{17},1^7,0^{41}$	$5^{12},4^4,3^{16},2^4,1^{22},0^{13}$
comp17.ectt	$\mathbf{10^2,6^2,4^5,2^{31},1^8,0^{22}}$	$\mathbf{10^2,6^2,5,4^{32},2^6,0^{27}}$	$10^2,6^2,4^8,3,2^{25},1^{12},0^{20}$	$10^2,6^2,5^9,4^{29},3^{16},2^7,1^5$
comp18.ectt	$6,4^{18},3,2^{15},1^4,0^{13}$	$\mathbf{7,6^{19},5^6,4^3,3^2,2^9,1^{11},0}$	$\mathbf{6,4^{17},3^3,2^{14},1^3,0^{14}}$	$7^4,5^5,4^{20},3^9,2^4,1^9,0$
comp19.ectt	$\mathbf{6^4,4^6,2^{12},1^5,0^{39}}$	$\mathbf{6^{14},5^9,4^7,2^{18},1^6,0^{12}}$	$6^4,4^7,2^{14},1^{10},0^{31}$	$7^2,6^4,5^{16},4^2,3^{19},2^4,1^{16},0^3$
comp20.ectt	$\mathbf{4^3,3^2,2^{27},1^{16},0^{30}}$	$\mathbf{5^2,4^{24},3^4,2,1,0^{46}}$	$4^5,3^2,2^{34},1^{13},0^{24}$	$5^{16},4^6,3^2,1^7,0^{47}$
comp21.ectt	$\mathbf{10,6^4,4^{15},3,2^{25},1^7,0^{25}}$	$10,6^{10},5^{33},4^4,2,1^6,0^{23}$	$10,6^4,5,4^{16},2^{27},1^4,0^{25}$	$\mathbf{10,6^9,5^{15},4^6,3,2^{31},0^{15}}$

as proposed in Section 3 improves the performance of the MAXMINFAIR_SA algorithm on most of the ITC2007 benchmark instances. Furthermore, we proposed a measure for quantifying how fair a timetable is with respect to max-min fairness. Using this measure helps to apply statistical methods in the analysis of the performance of randomized optimization algorithms for optimization problems with a bottleneck objective. In particular, it enables us to compare the average solution quality of the two variants of the MAXMINFAIR_SA algorithm.

References

1. Ruggero Bellio, Luca DiGaspero, and Andrea Schaerf. Design and statistical analysis of a hybrid local search algorithm for course timetabling. *Journal of Scheduling*, 15:49–61, 2012.
2. J. F. Benders. Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.
3. Dimitris Bertsimas, Vivek F. Farias, and Nikolaos Trichakis. The price of fairness. *Operations Research*, 59(1):17–31, 2011.
4. R. E. Burkard and F. Rendl. Lexicographic bottleneck problems. *Operations Research Letters*, 10:303–308, 1991.
5. Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.
6. Rainer Ernst Burkard and Uwe Zimmermann. *Weakly admissible transformations for solving algebraic assignment and transportation problems*, volume 12 of *Mathematical Progamming Study*, chapter 1, pages 1–18. North-Holland, 1980.
7. Edmund K. Burke, Jakub Marecek, Andrew J. Parkes, and Hana Rudová. Decomposition, reformulation, and diving in university course timetabling. *Computers & OR*, 37(3):582–597, 2010.
8. Edmund K. Burke, Jakub Mareček, Andrew J. Parkes, and Hana Rudová. A branch-and-cut procedure for the Udine course timetabling problem. *Annals of Operations Research*, 194(1):71–87, 2011.
9. F. Della Croce, V. Th. Paschos, and A. Tsoukias. An improved general procedure for lexicographic bottleneck problems. *Operations Research Letters*, 24:187–194, 1999.
10. Luca Di Gaspero, Barry McCollum, and Andrea Schaerf. The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (Track 3). Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1.0/1, School of Electronics, Electrical Engineering and Computer Science, Queens University, Belfast (UK), August 2007.
11. 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(1):3 – 27, 2004.
12. Allan M. Feldman and Roberto Serrano. *Welfare economics and social choice theory*. Springer, New York, NY, 2nd edition, 2006.
13. H. Gabow and R. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.
14. Iiro Harjunkoski and Ignacio E. Grossmann. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers & Chemical Engineering*, 26(11):1533 – 1552, 2002.
15. Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical report, Digital Equipment Corporation, September 1984.
16. Stacy L. Janak, Christodoulos A. Floudas, Josef Kallrath, and Norbert Vormbrock. Production scheduling of a large-scale industrial batch plant. i. short-term and medium-term scheduling. *Industrial & Engineering Chemistry Research*, 45(25):8234–8252, 2006.
17. Scott Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
18. Jon Kleinberg, Yuval Rabani, and Éva Tardos. Fairness in routing and load balancing. *Journal of Computer and System Sciences*, 63(1):2–20, 2001.
19. Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.
20. A. Kumar and J. Kleinberg. Fairness measures for resource allocation. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 75–85, 2000.
21. Gerald Lach and Marco E. Lübbecke. Curriculum based course timetabling: new solutions to Udine benchmark instances. *Annals of Operations Research*, 194:255–272, 2012.

22. Tian Lan, David Kao, Mung Chiang, and Ashutosh Sabharwal. An axiomatic theory of fairness in network resource allocation. In *INFOCOM*, pages 1343–1351. IEEE, 2010.
23. Zhipeng Lü and Jin-Kao Hao. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, 200(1):235–244, 2010.
24. Zhipeng Lü and Jin-Kao Hao. Adaptive Tabu Search for course timetabling. *European Journal of Operational Research*, 200(1):235–244, 2010.
25. Barry McCollum, Andrea Schaerf, Ben Paechter, Paul McMullan, Rhyd Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, and Edmund K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22:120–130, 2010.
26. Moritz Mühlenthaler and Rolf Wanka. Fairness in academic course timetabling. In *Proc. 9th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT)*, pages 114–130, 2012.
27. Tomáš Müller. ITC2007 solver description: A hybrid approach. *Annals of Operations Research*, 172(1):429–446, 2009.
28. Włodzimierz Ogryczak. Bicriteria models for fair and efficient resource allocation. In *Proc. 2nd International Conference on Social Informatics (SocInfo)*, pages 140–159. Springer, 2010.
29. Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization; Algorithms and Complexity*. Dover Publications, 1998.
30. David W. Pentico. Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774 – 793, 2007.
31. Celso C. Ribeiro and Sebastián Urrutia. Scheduling the brazilian soccer tournament with fairness and broadcast objectives. In *Proc. 6th Int. Conf. on Practice and Theory of Automated Timetabling*, PATAT, pages 147–157, Berlin, Heidelberg, 2007. Springer-Verlag.
32. Tomáaš Müller and Hana Rudová. Real-life curriculum-based timetabling. In *Proc. 9th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT)*, pages 57–72, 2012.
33. Pieter Smet, Simon Martin, Djamila Ouelhadj, Ender Özcan, and Greet Vanden Berghe. Investigation of fairness measures for nurse rostering. In *Proc. 9th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT)*, pages 369–372, 2012.
34. M. J. Soomer and G. M. Koole. Fairness in the aircraft landing problem. In *Proceedings of the Anna Valicek Competition 2008*, 2008.
35. Pamela H. Vance, Cynthia Barnhart, Ellis L. Johnson, and George L. Nemhauser. Airline Crew Scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45(2), M 1997.

MISTA 2013

A simulated annealing approach to the curriculum-based course timetabling problem

Ruggero Bellio · Sara Ceschia · Luca Di Gaspero · Andrea Schaerf · Tommaso Urli

1 Introduction

Course timetabling is a typical problem that all universities around the world have to face every semester. There exist a number of variants of this problem, depending on the specific requirements of the institution involved [12].

Thanks also to the international timetabling competition ITC-2007 [10], two formulations have, to some extent, lifted up to the status of "standard". These are the so-called Post-Enrolment Course Timetabling (PE-CTT) [7] and Curriculum-Based Course Timetabling (CB-CTT). The distinguishing difference between the two is the origin of the conflict matrix between courses, which is based on student enrolments and on predefined curricula published by the university, respectively. This is however only one of the differences, which actually include also many other features and cost components. Indeed, in the PE-CTT each event is self-standing, whereas in the CB-CTT a curriculum is composed by different courses that consist of multiple lectures and share the same students. In addition, the soft constraints are diverse: in the PE-CTT they are all related to events, penalising late, consecutive and isolated ones; in the CB-CTT they mainly involve curricula and courses, penalising isolated lectures in a curriculum, trying to minimise the number of days in which lectures of the same course are spread and the room transfers.

In this work we focus on CB-CTT, building upon our previous work on this problem [3]. Some of the ideas, however, are transferred from work on PE-CTT [5], on which we have obtained good results, including many new best-known solutions, applying a similar approach.

R. Bellio
DIES, University of Udine
via Tomadini, 30/A
E-mail: ruggero.bellio@uniud.it

S. Ceschia, L. Di Gaspero, A. Schaerf, and T. Urli
DIEGM, University of Udine
Via delle Scienze 206
E-mail: {sara.ceschia, l.digaspero, schaerf, tommaso.urli}@uniud.it

2 Search method

We propose a solution method based on Simulated Annealing (SA), along the lines developed for the PE-CTT [5]. The main modelling difference is that, in the CB-CTT formulation, the classrooms play an important role for the cost of a solution (mainly due to the RoomStability cost component which states that all lectures of the same course should be given in the same room), whereas in the PE-CTT all suitable rooms can be used interchangeably for the same lecture.

This observation has an impact on the definition of the neighbourhood, which is the one crucial feature of all methods based on local search. In fact, while for PE-CTT a move could consider only the period of a lecture, leaving the room implicit to be selected a-posteriori, for CB-CTT the room is one of the attributes of the move.

A main difference with respect to previous work on the PE-CTT, is the use of a non-geometric cooling scheme for SA. In this version, the temperature decreases faster in the beginning of the search and slower towards the end, in order to speed-up the process at high temperatures. This allows to "save" iterations that can be put to better use in the final steps of the search, when intensification becomes more important. This technique, already described by Johnson *et al.* [6] as *cutoffs*, is achieved by triggering the cooling steps not only based on the number of iterations expired but also on the number of number of moves accepted.

We do not set a minimum temperature, but we base our stop criterion on the total number of iterations. This way, we have a running time which is fixed to a predefined value, so that we can have a fair comparison with previous work.

3 Experimental analysis and results

We have tuned the main parameters of our method on a set of instances, generated by Lopes and Smith-Miles [8], designed to mimic the features of real-world instances from the University of Udine.

Competing parameter assignments were compared against each other using an *F-Race(RSD)* approach [4] where the values for the starting parameter configurations were sampled from the Hammersley point set [14]. This choice has been driven by two properties of this sequence that make it favourable to parameter tuning. First, points from the Hammersley set exhibit *low discrepancy*, i.e. they are well spread across the parameter space despite being random-like. Second, the sequence is *scalable* both with respect to the number of points to be sampled and to the number of dimensions of the sampling space. This allows to choose the number of racing configurations in advance, avoiding an full-factorial approach, while still tuning several parameters at once. Note that this process is generally applicable to any search method involving multiple parameters, and has been automated through a tool [13] for experiment design and analysis developed and open-sourced by our research group.

All the experiments were executed on an Ubuntu Linux 12.04 machine with 16 Intel® Xeon® CPU E5-2660 (2.20GHz) cores. The running time has been set to the one prescribed by the ICT-2007 (6 minutes on our machine) which, converted to number of iterations, is equal to $3.6 \cdot 10^8$.

The best parameter configuration has been tested on the `comp` instance set, which has been used extensively in the literature. Table 1 shows the results in comparison with the best available ones. The table shows that our SA solver obtains very good

Inst.	Müller ITC 07	Müller [11]	Lü & Hao [9]		Abdullah *et al.* [1]		Asín Achá & Nieuwenhuis [2]	SA		Best known
	avg	best	avg	best	avg	best	z	avg	best	
comp01	**5.0**	**5**	**5.0**	**5**	**5.0**	**5**	5	**5.00**	**5**	5
comp02	61.3	43	60.6	**34**	53.90	39	24	**53.0**	40	24
comp03	94.8	72	86.6	**70**	84.20	76	111	**79.0**	**70**	66
comp04	42.8	**35**	47.9	38	51.90	**35**	35	**38.3**	**35**	35
comp05	343.5	**298**	**328.5**	**298**	339.5	315	1343	365.20	326	290
comp06	56.8	**41**	69.9	47	64.40	50	27	**50.4**	**41**	27
comp07	33.9	14	28.2	19	**20.20**	**12**	6	23.8	17	6
comp08	46.5	39	51.4	43	47.90	**37**	37	**43.6**	40	37
comp09	113.1	103	113.2	99	113.90	104	171	**105.0**	**98**	96
comp10	21.3	**9**	38.0	16	24.10	10	4	**20.5**	11	4
comp11	**0.0**	**0**	**0.0**	**0**	**0.0**	**0**	0	**0.00**	**0**	0
comp12	351.6	331	365.0	**320**	355.90	337	977	**340.5**	325	300
comp13	73.9	66	76.2	65	72.40	**61**	59	**71.3**	64	59
comp14	61.8	**53**	62.9	52	63.30	**53**	51	**57.9**	54	51
comp15	94.8	–	87.8	**69**	88.00	73	111	**78.8**	70	66
comp16	41.2	–	53.7	38	51.70	32	18	**34.8**	**27**	18
comp17	86.6	–	100.5	80	86.20	72	56	**75.7**	**67**	56
comp18	91.7	–	82.6	**67**	85.80	77	83	**80.8**	69	62
comp19	68.8	–	75.0	**59**	78.10	60	57	**67.0**	61	57
comp20	**34.3**	–	58.2	35	42.90	**22**	4	38.8	33	4
comp21	108.0	–	125.3	105	121.50	95	86	**100.1**	**89**	75
	87.2	–	91.2	74.2	88.13	74.52	155.48	**82.3**	**73.4**	63.7

Table 1 Comparison of results

results, despite being a rather simple method, but with an extensive and principled tuning phase.

We have highlighted in bold the lowest average and best costs. The results labeled with z (Asìn Achà and Nieuwenhuis [2]) have not been compared with the others as they are obtained through a Max-SAT solver, by allotting a much higher runtime (from 10'000 to 100'000 seconds) than the one allowed in the ITC competition.

4 Conclusions and Future Work

In this paper we have presented an ongoing study on the effectiveness of a simple but properly tuned SA variant on a popular scheduling problem, namely the Curriculum-Based Course Timetabling. Our preliminary results are encouraging, comparing favorably with the current bests and suggesting that parameter tuning should be carried out in a more statistically principled way, in light of the major role it plays in solvers' performance. This is, of course, not only limited to SA as all meta-heuristics are, to various extents, sensitive to parameter values. For this reason we plan to carry out similar investigations starting with approaches similar to SA, such as Great Deluge, and moving on to more different meta-heuristics, such as Tabu-Search.

References

1. Salwani Abdullah, Hamza Turabieh, Barry McCollum, and Paul McMullan. A hybrid metaheuristic approach to the university course timetabling problem. *Journal of Heuristics*, 18(1):1–23, December 2010.
2. Roberto Asín Achá and Robert Nieuwenhuis. Curriculum-based course timetabling with SAT and MaxSAT. *Annals of Operations Research*, February 2012.
3. Ruggero Bellio, Luca Di Gaspero, and Andrea Schaerf. Design and statistical analysis of a hybrid local search algorithm for course timetabling. *Journal of Scheduling*, 15(1):49–61, 2012.

4. Mauro Birattari, Z. Yuan, P. Balaprakash, and Thomas Stützle. *F-Race and iterated F-race: An overview*. Springer, Berlin, 2010.
5. Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. *Computers & Operations Research*, 39:1615–1624, 2012.
6. D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operations Research*, 37(6):865–892, 1989.
7. Rhyd Lewis, Ben Paechter, and Barry McCollum. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Technical report, Cardiff University, Wales, UK, 2007.
8. Leo Lopes and Kate Smith-Miles. Pitfalls in instance generation for Udine timetabling. In *Proceedings of the LION4 Conference*, 2010.
9. Zhipeng Lü and Jin-Kao Hao. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, 200(1):235 – 244, 2009.
10. Barry McCollum, Andrea Schaerf, Ben Paechter, Paul McMullan, Rhyd Lewis, Andrew J. Parkes, Luca Di Gaspero, Rong Qu, , and Edmund K. Burke. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, 22(1):120–130, 2010.
11. Tomáš Müller. ITC2007 solver description: a hybrid approach. *Annals of Operations Research*, 172(1):429–446, 2009.
12. Andrea Schaerf. A survey of automated timetabling. *Artificial Intelligence Review*, 13(2):87–127, 1999.
13. Tommaso Urli. json2run: a tool for experiment design & analysis. *arXiv preprint arXiv:1305.1112*, 2013.
14. Tien-Tsin Wong, Wai-Shing Luk, and Pheng-Ann Heng. Sampling with Hammersley and Halton Points. *Journal of Graphics Tools*, 2(2):9–24, January 1997.

MISTA 2013

Bandit Algorithms using Monte Carlo Rollouts for Job Shop Scheduling

Einar Geirsson · Thomas Philip Runarsson

Abstract Various bandit algorithms for approximating solutions to the job-shop scheduling problem are presented. The methods use a one step look-ahead procedure to evaluate all jobs available for dispatching, by generating multiple feasible schedules via Monte Carlo rollouts. Jobs are then dispatched based on statistics collected from these rollouts. These different bandit algorithms differ in the way jobs are dispatched based on the statistics collected. Rollout algorithms designed for combinatorial optimization apply sequentially consistent heuristics, the algorithms presented here use inconsistent heuristics. The methods are tested on 600 job-shop problems of three different dimensions.

1 Introduction

Scheduling is the task of allocating a number of jobs to a number of machines to meet a predefined criteria. Due to its importance in production management, scheduling has been widely researched since the industrial revolution. The main difficulty scheduling problems such as the job-shop problem is that as more machines and jobs are added to the problem its complexity increases exponentially and so the computational time required to solve it. For this reason heuristic methods for finding approximate solutions have been proposed, ranging from very simple to complex and sophisticated methods. Typically a good approximation technique will be tailored for the particular scheduling task.

The approximate solution techniques described here are so called rollout algorithms. Rollout algorithms have previously been used by [1] and [2] for solving combinatorial optimization problems and [3] for the job-shop problem. A rollout implies generating a complete schedule from a partially built one using a *pilot* heuristic, or a number of pilot heuristics. Based on these rollouts a tree, not unlike a game tree, is constructed.

Einar Geirsson
University of Iceland
E-mail: einarge@gmail.com

Thomas Philip Runarsson
University of Iceland
E-mail: tpr@hi.is

This tree is then used to determine which job to dispatch next. In other words, the job-shop problem is formulated as a sequential decision problem. This approach has many similarities to Monte Carlo tree search (MCTS), which has over the recent years become popular in finding game playing strategies [4]. Recently, the performance of MCTS for job-shop scheduling was investigated in [5]. Here we investigate the idea of using Monte Carlo rollouts for job-scheduling further, but this time without constructing a tree and instead comparing simple bandit algorithms. Three different bandits are considered, the one that maximizes the mean payoff, one the maximizes the maximum payoff and a new approach we will call maximizing the mean quantile payoff.

One of the greatest difficulties of designing approximate or heuristic algorithms for the job-shop scheduling, is that different problem instances vary significantly and so does the performance of these algorithms. For our rollouts we will not use a problem tailored pilot heuristic to generate schedules. Instead, schedules will be generated purely at random. This relieves us of the design of problem specific heuristics for new scheduling problems and will produce feasible schedules with little computational effort. This is in essence similar to the approach taken in Monte Carlo computer Go.

The paper is organized as follows. First we formally define the job shop scheduling problem. This is followed by a description of the bandit problem, where the three different statistics which will be used to dispatch jobs are presented. Then the job-shop bandit algorithm is formally presented. Experimental results are then given for some 600 scheduling problems. The paper concludes with a discussion and main conclusions.

2 Job-Shop Scheduling

The job-shop problem consists of a set of jobs that need to be processed on a set of machines. The problem assumes that a machine can only process one job at a time and that the processing of a job, referred to as an operation, cannot be interrupted. In an $n \times m$ job-shop problem, n jobs must be processed on m machines [6]. The jobs are scheduled as a chain of operations where each operation needs to be processed during a given time period on each machine. The objective is to find the set of operations (a schedule) that has minimum length [7].

A set $\mathcal{M}$ of m machines is given together with a set $\mathcal{J}$ of n jobs and a set $\mathcal{O}$ of $n \times m$ operations. For each operation, $o \in \mathcal{O}$, there is a processing time, $p(o) \in \mathbb{N}$, a unique machine $M(o) \in \mathcal{M}$ where it is processed, and a unique job $J(o) \in \mathcal{J}$ to which it belongs. Hence each $o \in \mathcal{O}$ can be denoted by o_{ji} where j is the job and i the machine belonging to this operation.

The order of the jobs on the machines is specified in a permutation matrix $\sigma_{j,i} \in \mathcal{M}$, where $\sigma_{j,i}$ is the i-th machine that needs to process job j. The matrix σ decides the total ordering of operations for each job. Let $x(o_{j,i})$ be the starting time of job j on machine i. A schedule is a function, $S : \mathcal{O} \to \mathbb{Z}^+$, that for each operation o defines a start time $x(o)$. A feasible schedule must satisfy the following constraints:

$$x(o_{j,i}) \geq 0 \qquad \forall j \in \mathcal{J} \;\; i \in \mathcal{M} \qquad (1)$$

$$x(o_{j,\sigma_{j,i}}) \geq x(o_{j,\sigma_{j,i-1}}) + p(o_{j,\sigma_{j,i-1}}) \qquad \forall j \in \mathcal{J} \;\; i \in \mathcal{M} \qquad (2)$$

$$x(o_{j,i}) \geq x(o_{k,i}) + p(o_{k,i}) \text{ or } x(o_{k,i}) + p(o_{k,i}) \leq x(o_{j,i}) \qquad (3)$$

$$\forall j, k \in \mathcal{J}, j \neq k, \;\; i \in \mathcal{M}$$

Constraint (1) states that the starting time of each job must be non-negative. Constraint (2) guarantees that all precedence constraints are satisfied. Constraint (3) makes sure that no machine can process two jobs at the same time. The length of a schedule, or the *makespan*, is the completion time of the last job and is defined as

$$C_{max} = \max_{o \in \mathcal{O}} x(o) + p(o) \tag{4}$$

The most widely researched job-shop problems are deterministic, where the processing time is considered to be known and without uncertainties. Additional time required during processing, such as set-up times, is often implemented in the model as additional processing time. This is of course a simplification, since the processing time of a job in the real world is always stochastic. Despite this simplification, solving deterministic problems is a useful tool in scheduling to give insight in the workings of the job-shop [8].

2.1 Heuristics

Heuristics are simple dispatch rules gained from experience and expertise. Most of these rules are dependent on the type of problem at hand and are in general not efficient for different types of problems. Each method dispatches a job, one at a time, based on the current state of all machines and jobs, to create a feasible and good schedule [9].

Popular dispatching rules include shortest processing time (SPT, the job with the shortest processing time is dispatched next), Most WorK Remaining (MWKR, the job with the longest total remaining processing time) and Least Operation Number (LOPN, the job with the least operation number). These rules are among the simplest ones, and are by no means optimal.

3 Bandit Problems

The K-armed bandit can be considered as a decision problem where one is faced with a hypothetical slot machine with K levers. Each lever gives a payoff coming from a stationary but unknown probability distribution [10]. The goal is to find out which lever to pull in order to maximize the expected cumulative payoff received over a series of n-trials. To solve the problem a balance must be found between exploration (pulling the lever to find out the reward) and exploitation (pulling the best known lever repeatedly) [11]. Alternatively, for the max K armed bandit problem the aim is to maximize the maximum payoff received from the levers.

Each lever has a payoff called the *true value* of that lever. If the value of each lever is known, solving the bandit problem is trivial and the lever with the highest value is always chosen. Since it is assumed that the distribution associated with each lever are not known with certainty, it is necessary to keep an estimation of the value of the different levers. The lever that has the greatest estimated value is called the *greedy* lever. If the greedy lever is chosen it is called an *exploitation* action since the current knowledge of the lever values are exploited. If a non-greedy lever is chosen it is called an *exploration* action since the environment is being explored to improve the estimates of the lever values. Whether it is better to exploit or explore depends on a number of factors such as the uncertainty of the lever values and the number of remaining pulls.

Due to the impossibility of exploring and exploiting at the same time one often refers to the *conflict* between exploration and exploitation [12].

3.1 Average Payoff

When solving the K-armed bandit the means by which the levers are evaluated is critical. The methods vary from very simple to complex methods. The most straightforward method to estimate the value of a lever is to calculate the mean reward received so far from that lever. If after t plays, lever i has been chosen n_i times, giving the payoffs $r_1, r_2, \ldots, r_{n_i}$ then

$$Q_t(i) = \frac{r_1 + r_2 + \ldots r_{n_i}}{n_i} \tag{5}$$

If $n_i = 0$, then $Q_t(i)$ is defined at some initial value, such as $Q_0(i) = 0$. This method is called the sample-average method.

The intuitive choice of a lever is to choose lever $i^* = \arg_i \max Q_t(i)$, this always exploits current knowledge of the levers and due to uncertainties in the estimated value of the lever this may not necessarily be a good choice. To take this into consideration, so-called ϵ-greedy methods, are commonly used where a non-greedy lever is chosen with probability ϵ. The choice of ϵ depends on the task at hand. If the variance of the reward is high, it is best to use a high value of ϵ, to be able to evaluate the levers true value accurately. If the variance is low, it is better to have a small degree of exploration (low value of ϵ) since our estimate of the true value becomes quite accurate.

The conflict between exploration and exploitation is critical. A balance has to be found between exploring the environment for gathering statistics and searching for profitable actions, while taking the best action as often as possible and keeping the computational time at a minimum. One such technique is the so called upper confidence bounds (UCB) approach [13], which has become very popular recently. There are also methods, such as Gittins allocation indices [14], that allocate trials optimally by considering the payoff to be independent. In this work we will estimate the average payoff of each lever by assigning an equal number of trials to each lever. That is, we will be purely exploratory.

3.2 Maximum Payoff

The max K-armed bandit is a version of the K-armed bandit were the goal is to maximize the maximum payoff received over a series of n trials. In any given trial it is possible to choose between K levers. This version of the bandit problem is useful when dealing with deterministic combinatorial optimization problems like the job-shop problem [11]. The aim is to allocate trials to find the decision which is the most likely to maximize a single payoff, such a optimizing the makespan. The objective can be stated in the following way:

$$\max\left(\max_{i=1}^{K}\left(\max_{j=1}^{n_i} R_j(D_i)\right)\right) \tag{6}$$

where $R_j(D_i)$ is the payoff for the j-th trial of lever i with distribution D_i [15]. In [15] it is assumed that the payoffs are drawn from the generalized extreme value distribution

(GEV) and they derive an exploration strategy for allocating trials to different levers. The exploration strategy then follows the following Boltzmann exploration distribution:

$$P(i) = \frac{\exp(R_i/T)}{\sum_{j=1}^{K} \exp(R_j/T)} \tag{7}$$

where R_i is an estimator of the expected maximum of a series of the trial levers, i and T is an exponentially decreasing temperature parameter. The aim is to increase the allocation of trials to the best levers double exponentially and to do this [15] propose to set $T = \exp(-j)$.

It can be difficult to find a suitable distribution for the payoffs and [11] assume that the payoff distribution does not belong to any specific parametric family and make no formal assumptions about the underlying distribution. Their algorithm is called Threshold Ascent. This algorithm works best when two criteria are satisfied:

1. There is a threshold, $t_{critical}$ such that for all $t > t_{critical}$ the lever most likely to yield a payoff greater than t is the lever most likely to yield a payoff greater than $t_{critical}$. This lever is called i^*.
2. As t increases beyond $t_{critical}$ the gap between the probability that lever i^* yields a payoff greater than t and the corresponding probability for other levers is growing.

The Threshold Ascent for solving the max bandit problem sets a threshold $t_{critical}$ that varies over time. Initially it is set to zero and whenever s payoffs have been received, that are greater than $t_{critical}$, the value of $t_{critical}$ is incremented. A pseudocode for Threshold Ascent can be seen in algorithm 1.

Algorithm 1 Threshold ascent

Input: Number of levers K, parameter s, parameter δ, a small number Δ.
Output: Payoff of the different levers.
$t_{critical} \leftarrow 0$
for all i ={1,2,. . . , K} **do**
  $n_i \leftarrow 0$
end for
for all rollouts **do**
  while number of payoffs greater than threshold $\geq s$ **do**
    $t_{critical} \leftarrow t_{critical} + \Delta$
  end while
  $i^* \leftarrow \arg\max U(\frac{S_i}{n_i}, n_i)$
  Pull arm i^* and receive payoff R
  $n_i \leftarrow n_i + 1$
end for

The different levers are $i = 1, 2, \ldots, K$ and n_i is the number of times lever i has been pulled, S_i is the amount of rewards received by arm i above $t_{critical}$ and:

$$U(\mu, n) = \begin{cases} \mu + \frac{\alpha + \sqrt{2n\mu\alpha + \alpha^2}}{n} & \text{if } n > 0; \\ \infty & \text{otherwise.} \end{cases} \tag{8}$$

where $\alpha = \ln(\frac{2nk}{\delta})$ and δ is a parameter. The parameter s controls the tradeoff between exploration and exploitation. It is the amount of payoffs the strategy is based on. If $s = 1$ then the policy is only dependent on how many pulls each lever has received and the lever that has had the least pulls so far is chosen. If $s \to \infty$ the policy equals the policy for the usual K-armed bandit problem [11].

3.3 Average Quantile Payoff

Instead of collecting all payoffs and evaluating their average we propose a method under which we discard all trials (and payoffs) except those in the upper quantile. Each arm j is pulled an equal number of times n and the value of each lever is set as

$$R(j) = \frac{r_1 + r_2 + \ldots r_{\lfloor n/q \rfloor}}{\lfloor n/q \rfloor} \tag{9}$$

where $r_1 \geq r_2 \geq r_3 \geq \ldots \geq r_n$. The result of this approach is a compromise between the K-armed bandit and the max K-armed bandit. That is, we would like to choose the lever that gives us regularly better payoff on average.

Instead of gathering statistics of all rollouts and evaluating their average we propose discarding all rollouts except those in the lower quantile (since we are considering a minimization problem). In the experimental study we will set the quantiles at either $q = 4$ or 8. After completion of the rollout stage, the quantile K-armed bandit algorithm dispatches the job that produced rollout schedules with the lowest average makespan in the lower quantile.

4 The Job-Shop Bandit

A rollout algorithm based on the job-shop problem, as presented by [5] and [3], uses the property that the problem can be defined as a sequential decision problem where the solution is built in stages. A partial solution is called a k-solution. The components, in this case the operations, are chosen one at a time to build a complete and feasible schedule.

The rollout algorithm for the job-shop problem uses a one step look-ahead procedure to choose the next job. For every k-solution the number of possible nodes to visit next are at most the same as the number of jobs. The Monte Carlo rollout generates a complete feasible solution from any given k-solution. This rollout simply assigns any feasible job for dispatching with equal probability. The node that gives the best value, corresponds to the next chosen operation. From this new $(k+1)$-solution the procedure is repeated until all nodes have been visited. A pseudocode for the bandit algorithm for solving the job-shop problem is given in algorithm 2. For the scheduling task each job must visit each machine once, for this reason the variable c_j records the number of machines job j has visited so far. This variable is then used to determine the set of feasible jobs to be dispatched.

Algorithm 2 Rollout algorithm for job-shop scheduling

Input: k-solution S_k, precedence matrix σ, J, number of machines m, $(c_1, \ldots, c_n)$ and rollout budget B (depends on the exploration policy used).
Output: A complete and feasible schedule.
repeat
  for all $j \in J$ **do**
    if $c_j \leq m$ **then**
      $Q_j = \text{Rollouts}(S_k, j, B_j)$ where Q_j denotes the value of dispatching job j.
    end if
  end for
  $j = \arg\min_j Q_j$, $\forall c_j \leq m$.
  $S_{k+1} \leftarrow S_k \oplus o_{j,\sigma_{j,c_j}}$ append operation to k-solution
  $k \leftarrow k + 1$
  $c_j \leftarrow c_j + 1$
until all operations have been assigned

Due to the stochastic nature of the rollouts different solutions will be constructed. By allocating a number of trials for every job, using known exploration and exploitation strategies, it is possible to generate several schedules originating from the same job. By considering the distribution of these schedules for each job, it is possible to evaluate each job and choose the one showing the most promising value. The four stages of the job-shop bandit are:

1. Find all feasible dispatches.
2. Allocate rollouts originating from these dispatches.
3. Evaluate schedules found during rollouts.
4. Dispatching the most promising job and return to stage one.

While dispatching jobs the best rollout solution is kept until a better one is found. This will be the solution returned by all bandit algorithms presented here.

5 Experimental Study

The performance of the proposed job-shop bandit algorithms are now tested in this section. Since the rollout heuristics are a purely randomised dispatching of jobs, the algorithms must be tested multiple times. Scheduling problems of different dimensions are also investigated in order to measure their scalability.

5.1 Experimental Setup

All algorithms are tested on 600 variations of the job-shop problem. These problems are divided equally into three different $n \times m$ sizes: 6×6, 10×10 and 14×14. The problems were created using the methodology proposed by [16] and solved to optimality using a branch and bound algorithm developed by [17]. The processing time of the jobs, on all the machines, for all problems is uniformly distributed between 1 and 200. The processing order of every job is a random permutation. Every job has to visit every machine once. The rollout algorithms were additionally tested on four well known 20×20 benchmark problems proposed by [18], where we will compare our results with the best known solution.

Every problem was solved with 6 different budgets for the rollouts: 100, 500, 1.000, 3.000, 5.000 and 10.000. A total of 30 independent trials were carried out for every problem and corresponding rollout budget. Each problem was, therefore, solved 180 times in all. In all cases each bandit algorithm will return the best k-solution found during its run.

5.2 Performance Criteria

When comparing the results of the different algorithms it is necessary to have an unbiased performance measure. Here the mean percentage error (MPE) and average mean percentage error (AMPE) are used

$$\mathrm{MPE}_j = \frac{1}{n}\sum_{i=1}^{n}\frac{R_i - O_j}{O_j} \tag{10}$$

$$\mathrm{AMPE} = \frac{1}{t}\sum_{j=1}^{t}\mathrm{MPE}_j \tag{11}$$

where j is the problem index, n is the number of trials, R_i is the solution found with rollout algorithm in the i-th trial and O_j is the optimal solution of problem instance j. For our experiments $t = 200$ and $n = 30$. The results are tested for statistical significance with the Wilcoxon rank sum test.

The following statistics will be displayed in tables below:

1. **Min:** The *minimum* MPE found for a problem.
2. **Mean:** *AMPE* of all solutions found.
3. **Max:** The *maximum* MPE found for a problem.
4. **Stdev:** *Standard deviation* of the solutions found.
5. **Opt:** The percentage of solutions found that were *optimal*.

These results are based on 10.000 rollouts for each job dispatch decision made, unless otherwise indicated. Furthermore, all values are displayed as percentages. The rollouts rely purely on randomly dispatching of legal jobs with equal probability. The results for the four different bandit algorithms and for three well known heuristics will now be presented.

5.3 Average versus Maximum Payoff

The results for the average payoff (APB) using a purely exploratory job-shop bandit is compared with using the threshold ascent (TAB) exploratory policy in table 1. Furthermore, fortified versions of this experiment is presented in table 2. In these cases the globally best found solution so far, found perhaps by a previous k-solution, is followed when no better solution was found. For the 6×6 and 10×10 problems the performance is inferior to the fortified rollout algorithm. However, for the 14×14 problems, the average rollout algorithm performs better than the fortified rollout algorithm on average.

Threshold ascent depends on two parameters, s and δ. The parameter s is the number of solutions that are kept and the next exploration move is chosen from. Experiments were made for the following values $s = 30, 50, 70, 100, 200$ and $\delta = 0.1$ as is done by [11]. Experiments using these parameter settings did not result in statistically different results. For this reason we present only the results for $s = 30$ and $\delta = 0.1$. The threshold ascent has the advantage for the smaller problems for the fortified algorithms, but a purely exploratory strategy, non-fortified, has the overall performance advantage for larger problems. The threshold ascent exploration strategy is best when used in the fortified setting since it explores jobs that give good schedules more often. In the non-fortified setting the uncertainties of the different job dispatches differ significantly and are therefore not reliable for decision making.

Table 1 Results for APB and TAB.

Problem	Method	min	mean	max	stdev	opt
6×6	APB	0.00	4.62	16.19	1.30	14.77
	TAB	1.32	6.71	17.46	3.84	8.10
10×10	APB	3.12	7.42	12.04	2.21	0.08
	TAB	4.99	10.42	15.38	3.19	0.05
14×14	APB	4.11	9.17	12.71	1.95	0.00
	TAB	6.20	13.29	17.22	2.67	0.18

Table 2 Results for the fortified versions of APB and TAB.

Problem	Method	min	mean	max	stdev	opt
6×6	FAPB	0.00	0.30	8.42	0.26	81.83
	FTAB	0.00	0.30	8.35	0.29	81.87
10×10	FAPB	0.84	4.92	9.58	1.76	0.57
	FTAB	0.59	4.84	9.17	1.85	0.58
14×14	FAPB	6.41	11.26	14.32	2.16	0.00
	FTAB	5.95	11.19	14.03	1.76	0.22

5.4 Average Quantile Payoff

The quantile purely exploratory bandit (AQB) algorithm follows the job that has the best average of solutions in the lower quantile and opposed to the overall best average. Only a non-fortified version of this algorithm will be considered. Two separate quantiles were tested, quartile ($q = 4$) and octile ($q = 8$), i.e. the ordered solutions were divided into 4 and 8 equally large subsets and the job with the lowest makespan average in the lowest quartile or octile was chosen to be processed next. Results for the quantile rollout algorithm are listed in table 3.

Table 3 Results for the quantile purely exploratory bandit.

Problem	Quantile	min	mean	max	stdev	opt
6×6	Octile	0.00	0.37	8.41	0.24	77.32
	Quartile	0.00	0.37	8.46	0.24	76.95
10×10	Octile	0.22	3.83	7.89	1.19	1.20
	Quartile	0.20	3.86	7.78	1.16	1.50
14×14	Octile	3.17	6.99	9.85	1.40	0.33
	Quartile	3.40	6.81	10.14	0.98	0.25

For the 6×6 and 10×10 problems there is no statistical difference between choosing the lower quartile or octile. For the 14×14 problems using the quartile gives slightly better results on average but the difference is still not statistically significant. The optimal solution, of the 14×14 problems, is only found in a few cases but the method produces results on average that are not very far off from the optimal solution. The performance of the AQBs is significantly better than the bandits discussed so far.

5.5 Results for Different Number of Rollouts

Rollouts were made using six different budgets of rollouts for all the methods: 100, 500, 1.000, 3.000, 5.000 and 10.000. Table 4, shows the average mean percentage error (AMPE) for all the methods and all the different problem sizes. As expected the AMPE decreases with increasing amount of rollouts. For the 6×6 problems the results do not improve significantly when increasing the rollout number above 3.000 rollouts. For the 10×10 problems there is an improvement going above 3.000 rollouts but the gain is limited. For the 14×14 problems, the gain is significant and better results might be obtained by going above 10.000 rollouts. The quantile bandit has an advantage once a budget of more than 500 is used.

Table 4 Average mean percentage error for different budget of rollouts.

		Number of rollouts					
Problem	Method	100	500	1,000	3,000	5,000	10,000
6×6	APB	6.45	5.14	4.90	4.71	4.67	4.62
	FAPB	2.27	1.16	0.87	0.49	0.38	0.30
	FTAB	2.24	1.13	0.86	0.49	0.40	0.30
	$AQB_{1/4}$	2.41	1.27	0.96	0.62	0.50	0.36
10×10	PEB	10.39	7.50	6.57	5.38	4.96	4.34
	FAPB	9.13	7.33	6.64	5.77	5.41	4.92
	FTAB	9.08	7.23	6.61	5.72	5.33	4.84
	$AQB_{1/4}$	10.73	6.55	5.64	4.50	4.16	3.83
14×14	APB	19.69	14.17	12.25	10.32	9.70	9.17
	FAPB	15.77	14.04	13.33	12.28	11.76	11.26
	FTAB	15.92	14.01	13.27	12.24	11.84	11.19
	$AQB_{1/4}$	18.43	11.91	10.17	8.07	7.47	6.93

5.6 Heuristics

The results of the heuristic dispatching rules discussed in section 2.1 are depicted in table 5. The three dispatching rules studied are MWKR, SPT and LOPN, showing that

MWKR and LOPN behave similarly and significantly outperform SPT. Further, the performances of SPT significantly decrease with the problem size, whereas MWKR and LOPN demonstrate an excellent scalability in the considered size range. All the bandit algorithms outperform the heuristic dispatching rules considerably for all problem sizes, even for a smaller number of rollouts.

Table 5 Results for the greedy algorithms.

Problem	Method	min	mean	max	stdev	opt
6×6	MWKR	0.00	15.50	38.40	8.40	1.00
	SPT	13.70	39.00	81.60	15.00	0.00
	LOPN	1.70	17.80	36.90	8.30	0.00
10×10	MWKR	9.60	22.80	43.00	6.90	0.00
	SPT	30.30	65.40	116.10	16.60	0.00
	LOPN	10.30	21.60	36.90	6.10	0.00
14×14	MWKR	15.90	26.40	39.90	5.20	0.00
	SPT	58.40	101.20	172.10	24.40	0.00
	LOPN	15.00	25.30	37.60	4.80	0.00

5.7 Yamada 20×20 Problems

In [18] four different 20×20 benchmark job-shop problems are given. No optimal solution has been proven for these problems and the results are compared to best known solutions given by [19]. The results are presented in table 6. It can be seen that the quantile average rollout algorithm produces the best results for all four problems. The same pattern was seen for the 14×14 problems in table 4. For these problems the non-fortified versions outperform the fortified bandit algorithms. When using the quantile rollout algorithm the quartile outperforms the octile. The difference between the two settings has increased compared to what was observed for the 14×14 problems in table 3.

6 Summary and Discussion

The bandit algorithms presented in this study show promising results, outperforming well known dispatching rules. Using simple exploration/exploitation tactics it is possible to generate good schedules for job-shop problems. For smaller problems, 6×6, the fortified rollout algorithm finds the optimal solution for most runs with a budget of 10.000 rollouts, especially with threshold ascent exploration policy. For the larger problems, 14×14 and 20×20, the rollout algorithms were able to generate good schedules in a reasonable time.

For these problem instances a purely exploratory algorithm outperformed the fortified rollout approach. This is interesting since the fortified rollout algorithm is sequentially improving. This can perhaps be explained by the fact that for larger problems the number of feasible solutions is extremely high and the probability of finding a good schedule early on is small. This may result in the fortified rollout algorithm placing too much emphasis on schedules found early on in the scheduling process. The non-fortified version on the other hand dispatches the job that finds the best schedules on average and therefore does not depend upon finding good schedules early on.

Table 6 Results for the 20×20 problems.

Problem	Method	min	mean	max	stdev	% best
yn01	APB	7.55	10.68	12.73	1.39	0.00
	FAPB	12.16	16.88	19.82	1.83	0.00
	FTAB	11.60	17.11	22.52	2.41	0.00
	$AQB_{1/4}$	6.76	8.48	10.59	0.95	0.00
	$AQB_{1/8}$	7.66	9.77	12.61	1.35	0.00
yn02	APB	8.14	11.20	14.19	1.45	0.00
	FAPB	12.65	17.95	22.00	1.90	0.00
	FTAB	13.64	19.03	22.22	1.84	0.00
	$AQB_{1/4}$	6.82	9.72	13.64	1.45	0.00
	$AQB_{1/8}$	7.92	10.72	13.53	1.43	0.00
yn03	APB	7.95	10.96	13.77	1.51	0.00
	FAPB	12.99	17.81	22.40	1.98	0.00
	FTAB	13.21	18.07	23.29	2.03	0.00
	$AQB_{1/4}$	6.05	9.00	11.09	1.21	0.00
	$AQB_{1/8}$	7.05	9.79	13.21	1.47	0.00
yn04	APB	9.50	11.48	14.67	1.13	0.00
	FAPB	12.60	17.59	22.31	2.03	0.00
	FTAB	13.53	17.01	21.90	1.92	0.00
	$AQB_{1/4}$	6.71	9.77	13.02	1.52	0.00
	$AQB_{1/8}$	6.10	10.62	12.81	1.44	0.00

The quantile rollout algorithm was the most successful rollout algorithm for solving larger job-shop problems. This method was also the most consistent one, producing good schedules for both the smaller as well as the larger problems. The advantage of the quantile rollout algorithm is that it dispatches jobs with a higher likelihood of finding good solutions. When a job is dispatched based on the average there may still be a greater probability of finding low quality solutions. Similarly, dispatches based on extreme values, such as the fortified rollout algorithm, may put too much bias on the best schedule found for the given set of rollouts, which in return may result in a low probability of visiting regions of greater solution quality.

Rollouts were made for six different rollout budgets: 100, 500, 1.000, 3.000, 5.000 and 10.000. There is no need for more than 3000 rollouts when solving smaller problem instances, but for larger problem instances (14×14 and 20×20) it might be possible to generate better schedules by increasing the rollout budget above 10.000.

The rollout algorithms give good performance results for deterministic scheduling problems, but the methodology is easily transferable to stochastic problems.

References

1. Bertsekas, D.P., Tsitsiklis, J.N., Wu, C.: Rollout algorithms for combinatorial optimization. Journal of Heuristics **3**, 245–262 (1997)
2. Duin, C., Voß, S.: The Pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs. Networks **34**(3), 181–191 (1999)
3. Meloni, C., Pacciarelli, D., Pranzo, M.: A rollout metaheuristic for job shop scheduling problems. Annals of Operations Research **131**(1), 215–235 (2004). URL http://www.springerlink.com/index/U057107807305517.pdf
4. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. Computational Intelligence and AI in Games, IEEE Transactions on **4**(1), 1–43 (2012)

5. Runarsson, T.P., Schoenauer, M., Sebag, M.: Pilot, Rollout and Monte Carlo Tree Search Methods for Job Shop Scheduling. In: Y. Hamadi (ed.) Learning and Intelligent Optimization, Lecture Notes in Computer Science. Springer Berlin, Heidelberg (2012)
6. Carlier, J., Pinson, E.: An algorithm for solving the job-shop problem. Management Science **35**(2), 164–176 (1989)
7. Vaessens, R.J.M., Aarts, E., Lenstra, J.: Job shop scheduling by local search. INFORMS Journal on Computing **8**, 302–317 (1994)
8. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3rd edn. Prentice Hall (2008)
9. Kawai, T., Fujimoto, Y.: An efficient combination of dispatch rules for job-shop scheduling problem. IEEE International Conference on Industrial Informatics pp. 484–488 (2005)
10. Alpaydin, E.: Introduction to Machine Learning, 2nd edn. The MIT Press (2010)
11. Streeter, M., Smith, S.: A simple distribution-free approach to the max k-armed bandit problem. Principles and Practice of Constraint Programming **4204**(2006), 560–574 (2006)
12. Sutton, R., Barto, A.: Reinforcement learning: an introduction. Adaptive computation and machine learning. MIT Press (1998)
13. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time Analysis of the Multiarmed Bandit Problem. Machine learning **47**(2), 235–256 (2002)
14. Gittins, J.: Bandit processes and dynamic allocation indices. Journal of the Royal Statistical Society. Series B (Methodological) **41**(2), 148–177 (1979). URL http://www.jstor.org/stable/2985029
15. Cicirello, V., Smith, S.: The max k-armed bandit: A new model of exploration applied to search heuristic selection. In: Proceedings of the National Conference on Artificial Intelligence, vol. 20. AAAI Press (2005)
16. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993)
17. Brucker, P.: Scheduling algorithms, 5th edn. Springer (2007)
18. Yamada, T., Nakano, R.: A genetic algorithm applicable to large-scale job shop Problems. In: Parallel Problem Solving from Nature, pp. 281–290 (1992)
19. Banharnsakun, A., Sirinaovakul, B., Achalakul, T.: Job Shop Scheduling with the Best-so-far ABC. Engineering Applications of Artificial Intelligence (2006), 1–11 (2011)

MISTA 2013

Late Acceptance Hill-Climbing Applied to the High School Timetabling Problem

George H.G. Fonseca, Haroldo G. Santos,
Túlio A.M. Toffolo

Abstract This work presents an application of the Late Acceptance Hill-Climbing (LAHC) algorithm and some variations to solve the High School Timetabling Problem. The addressed model of the problem was proposed by the Third International Timetabling Competition (ITC2012), which released many real instances from educational institutions around the world and provided mechanisms to fairly compare the performance of solvers for the problem. The LAHC algorithm was able to overcome the solver proposed by the winner of the ITC2012, which consisted in a Simulated Annealing and Iterated local Search hybrid approach. Some variants of the LAHC algorithms showed promissory results. The Stagnation Free version of LAHC, for example, improved the results of the original algorithm. The combination of Simulated Annealing and LAHC was also a very promising alternative, and obtained the best average solutions for the considered instances. These results, coupled with another reports in the literature, points out that the LAHC is a reliable method that can compete with widely known local search algorithms.

Keywords Late Acceptance Hill-Climbing · Third International Timetabling Competition · High School Timetabling Problem · Local Search

1 Introduction

The High School Timetabling Problem is faced by many educational institutions around the world. It basically consists in assigning teachers and classes to timeslots, in such way that no teacher or class attends more than one lesson at same time. Generally, this assignment is repeated weekly until the end of the semester. Many other constraints can be considered as well, like the availability of teachers and the limit of lessons on the same subject in a day. Beyond its practical importance, this problem is $\mathcal{NP}$-Hard

George H. G. Fosneca
Exacts and Applied Sciences Department, Federal University of Ouro Preto, Brazil
E-mail: george@decea.ufop.br

Haroldo G. Santos and Túlio A. M. Toffolo
Computing Department, Federal University of Ouro Preto, Brazil
E-mail: haroldo@iceb.ufop.br, tulio@toffolo.com.br

[5] and progress in solving such problems is a major goal of current researches in Operations Research and Artificial Intelligence.

Due to the problem's importance, three international competitions were organized to bring scientists with different approaches to compare their techniques in the same environment. The first one happened in 2003 [7] and was won by [13] with a 3-phase Simulated Annealing-based [12] algorithm. The second one [14] was held in 2007 and, composed by three separated tracks, was won by [15] with a variation of the Simulated Annealing algorithm. The last one [14] happened in 2012 and was won by [4] with a Simulated Annealing and Iterated Local Search hybrid approach.

As the competitions pointed, local search methods are achieving the best results to the High School Timetabling Problem. In special, the Simulated Annealing metaheuristic, which composed the solver of all winners. Methods based in Integer Programming were proposed to the problem [20], but they can only solve exactly a subset of instances of the problem in a viable processing time.

This paper aims to present a new technique applied to the Third International Timetabling Completions' model of High School Timetabling Problem. It is based on a local search method called Late Acceptance Hill-Climbing (LAHC), proposed by [3]. The method obtained exciting results, outperforming the best results from the Third ITC's. Moreover, it is easy to implement and we believe that can be extended to many others optimization problems.

The remaining of this work is organized as follows. In section 2 the model of High School Timetabling Problem proposed by the Third ITC is presented. Section 3 presents the solution approach. Section 4 presents the computational experiments and results and, finally, in Section 5 the concluding remarks are presented.

2 High School Timetabling Problem Model

The Third ITC aimed to stimulate the timetabling research generally, and especially to encourage the alignment of research with practice by offering real-world instances of timetabling problems. The organizers also provided a benchmark to adjust processing times and a validator of solutions.

The instances were specified in the XHSTT format, which is an XML based format, strictly made to specify timetabling problems. [19] also highlighted that this format can specify instances of another timetabling problems besides the scholar context. Thus, our solver is able to solve any instance of timetabling problem specified in this format.

The addressed model of High School Timetabling (HST) Problem came up with the goal of providing a generic model capable to address the various features of the High School Timetabling Problem around the world [8, 23, 16, 21, 6, 20, 18]. The model is split in four main entities.

2.1 Times

The time entity consists of a single timeslot or a set of timeslots, called time group. The timeslots are commonly also grouped by day (e.g. timeslots of Monday).

2.2 Resources

The resources are divided in three main categories: classes, teachers and rooms [18]:

classes a group of students attends to an event (lesson). Important constraints to the students are control their idle times and the number of lesson by day;
teachers a teacher can be pre-assigned to attend an event. In some cases it is not pre-assigned and should be assigned according to their qualifications and workload limits;
rooms most of events take place on a room. One room has a certain capacity and a set of features.

2.3 Events

An event is a set of meets (lessons) about a subject. Events may be grouped into an event group. A timeslot assignment to an event is called a meet and a resource assignment to an event is called task. The term class is used to designate a group of students who attend the same set of events. Other kinds of events, like meetings, are allowed by the model [18]. An event has the following attributes:

- a *duration* that represents the number of lesson which an event is composed;
- a *course* related to the event;
- the *workload* that will be added to the total workload of resources assigned to the event;
- the *pre-assigned resources* to attend the event (optional);
- the *pre-assigned timeslots* to attend the event (optional).

2.4 Constraints

Post *et al.* [18] groups the constraints in three categories: basic constraints of scheduling, constraints of events and constraints of resources. The objective function $f(.)$ is calculated in terms of violations of the constraints. These violations are penalized according to weights (so this is a minimization problem). They are also divided in hard constraints, whose attendance is mandatory; and soft, whose attendance is desirable but not mandatory. Each instance can define whether a constraint is hard or soft and its weight. For more details, see [18].

2.4.1 Basic Constraints of Scheduling

- Assign Time. Assign the required number of timeslots to each lesson;
- Assign Resource. Assign the required resources to each lesson;
- Prefer Times. Indicates that some event has preference for a particular timeslot;
- Prefer Resources. Indicates that some event has preference for a particular resource.

2.4.2 Constraints to Events

- Link Events. Schedule a set of events to the same timeslot;
- Spread Events. Specify the allowed number of occurrences of an event in time groups between a minimum a maximum number of times. This constraint can be used, for example, to define a daily limit of lessons;
- Avoid Split Assignments. For each event, assign a particular resource to all of his meets. With this constraint, we can enforce all the allocations of an event to lie in the same room;
- Distribute Split Events. For each event, assign between a minimum and a maximum meets of a given duration. This constraint justify itself since a great number of consecutive lessons of the same subject can prejudice the students' performance;
- Split Events. Limit the number of non-consecutive meets that an event should be scheduled and his duration. One example of application of this constraint is to enforce an event composed of four lessons to be assigned in two blocks of contiguous lessons. This may be useful when a single lesson is not enough to discuss about a subject.

2.4.3 Constraints to Resources

- Avoid Clashes. Assign the resources without clashes (i.e. without assign the same resource to more than one lesson at any timeslot);
- Avoid Unavailable Times. Avoid assigning resources on the times that are not available. With this constraint, it is possible, for example, to avoid that a room is used in a timeslot when it was reserved to be cleaned or to avoid that a teacher is allocated in a timeslot in which he attends to a class as a student in another course;
- Limit Workload Constraint. To each event there is an associated workload. This workload is added to the resources which attend the event. The goal of this constraint is to keep the workload of the resources between a minimum and a maximum bound;
- Limit Idle Times. The number of idle times in each time group should lie between a minimum and a maximum bound for each resource; typically, a time group consists of all timeslots of a given day of the week. this constraint is used to avoid times inactive timeslots between active timeslots in the schedule of each resource;
- Limit Busy Times. The number of busy times in each day should lie between a minimum and a maximum bound for each resource. A great number of allocations in the same day can prejudice the students and teachers performance;
- Cluster Busy Times. The number of time groups with a timeslot assigned to a resource should lie between a minimum and a maximum limit; this can be used, for example, to concentrate teacher's activities in a as few days as possible.

3 Solution Approach

Our approach uses the Kingston High School Timetabling Engine (KHE), by [10], to generate an initial solution. Afterwards, the metaheuristic Late Acceptance Hill-Climbing (LAHC) or one of its variations perform local search around this solution to improve it. These elements will be explained in the following subsections.

3.1 Build Method

The KHE engine is a platform for handling instances of the addressed problem. It also provides a solver, used to build initial solutions in the presented approach. The KHE's solver was chosen to generate the initial solutions since it is able to find good initial solutions efficiently (see Table 5).

The incorporated solver is based on the concept of Hierarchical Timetabling [9], where smaller allocations are joint to generate bigger blocks of allocation until a full representation of the solution is developed. Hierarchical Timetabling is supported by the Layer Tree data structure [9], consisting of nodes that represent the required meet and task allocation. An allocation may appear in at most one node. A Layer is a subset of nodes having the propriety that none of them can be overlapped in time. Commonly, nodes are grouped in a Layer when they share resources.

The hard constraints of the problem are modeled with this data structure and then a Matching problem is solved to find the times/resources allocations. The Matching is done by connecting each node to a timeslot or resource, respecting the property of its Layer. For further details, see [9] and [10].

3.2 Neighborhood Structure

The neighborhood structure $N(s)$ is composed by six movements. The neighborhood about only one movement k is denoted by $N_k(s)$. The movements are:

1. Lesson Swap (LS). Two lessons l_1 and l_2 are selected and have their timeslots t_1 and t_2 swapped;
2. Lesson Move (LM). A lesson l_1 is moved from its original timeslot t_1 to a new timeslot t_2;
3. Lesson Block Move (LBM). As LS movement, swap the timeslot of two lessons l_1 and l_2, but, when the lessons have different duration, l_1 is moved to the last timeslot occupied by l_2. This movement allow timeslot swaps without lose the contiguity of allocations;
4. Resource Swap (RS). Two lessons e_1 and e_2 have their assigned resources r_1 and r_2 swapped. Resources r_1 and r_2 should play the same role to allow the swap (e.g. both have to be teachers);
5. Resource Move (RM). A lesson l_1 has his assigned resource r_1 replaced by a new resource r_2 between the available resources to attend it;
6. Kempe Move (KM). Two timeslots t_1 and t_2 are selected. Every lesson of t_1 and t_2 are listed and represent a node n in a graph. If two nodes (lessons) n_1 and n_2 in this graph share resources, they are connected with an edge. Edges are created only between nodes assigned in distinct timeslots, thus, the generated graph is a bipartite one, known as conflict graph. Every edge in the conflict graph also has a weight, formed by the cost difference in the objective function assuming the exchange of timeslots between the events depicted in the pair (n_1, n_2). Afterwards, we look up for the path with the lowest cost in the conflict graph and makes up the exchange of timeslots in chain;

The movement k in $N(s)$ is randomly selected in order to generate a neighbor. If the instance require the assignment of resources (i.e. there exists at least one Assign Resource constraint), the movements are chosen based on the following probabilities:

ES = 0.20, EM = 0.38, EBM = 0.10, RS = 0.20, RM = 0.10 and KM = 0.02. Otherwise, the movements RS and RM are not used and the odds become: ES = 0.40, EM = 0.38, EBS = 0.20 and KM = 0.02. These values were empirically adjusted.

3.3 Late Acceptance Hill-Climbing

Late Acceptance Hill-Climbing metaheuristic was recently proposed by [2]. It consists in an adaption of the classical Hill-Climbing method. It relies on comparing a new candidate solution with the last l solutions considered in the past before accepting or rejecting it. Note that the candidate solution may be accepted even if it is worse than the current solution, since it is compared to the current solution at the i-th (past) iteration.

This metaheuristic was created with three goals in mind: to be a one-point search procedure that does not employ an artificial cooling schedule like Simulated Annealing does; to effectively use the information collected during previous iterations of the search; and to employ a simple acceptance mechanism (i.e. almost as simple as Hill-Climbing) [3].

In this method, we store a list $f'_k = f_0, f_1, ..., f_{l-1}$ of fitness of solutions. Initially this list is filled with the cost of the initial solution s: $f'_k \leftarrow f(s)\ \forall k \in \{0, ..., l-1\}$. At each iteration i, a candidate solution s' is generated. The candidate solution is accepted if its fitness is smaller or equals to the fitness stored at i mod l position of f'. Moreover, if this solution is better than the best solution found, it is updated: $s^* \leftarrow s'$. Afterwards, the position $v = i$ mod l of f' is updated: $f'_v \leftarrow f(s')$. This process repeats until a stop condition is reached. The Algorithm 1 presents our implementation of LAHC. Note that the adopted stop condition is a timeout. This decision is discussed in Section 4. Some successful examples of application of LAHC can also be found at [22], [1] and [17].

Algorithm 1: Developed implementation of LAHC

```
Input: Initial solution s and parameter l.
Output: Best solution s* found.
f'_k ← f(s) ∀k ∈ {0, ..., l - 1};
s* ← s;
i ← 0;
while elapsedTime < timeout do
    Generate a random neighbor s'∈N(s);
    v ← i mod l;
    if f(s') ≤ f'_v then
        s ← s';
        if f(s) < f(s*) then
            s* ← s;
    f'_v ← f(s);
    i ← i + 1;
return s*;
```

3.3.1 Variations of LAHC

The LAHC metaheuristic was not fully explored yet, thus, lots of modifications and hybrid algorithms may be experimented in order to improve its effectiveness [3]. The following subsections presents two proposed variations of LAHC. Their combination with the Simulated Annealing algorithm was also explored in this work.

3.3.2 Stagnation Free LAHC

The LAHC method may get stuck in local optima, even with high sizes of list of fitness. In the late phase of its application it's hard to improve the current solution, thus the algorithm may store the same fitness in all of its positions and avoid the acceptance of worsening movements. To fix this problem, we propose a Stagnation Free version of LAHC, the sf-LAHC method.

In the sf-LAHC method we reheat the algorithm when it reaches a stagnation condition. In our implementation, the reheat consists in retrieve the list of fitness in the last time in which it was improved, f''_k. Thus, lots of worsening movements may be accepted again. The stagnation condition is determined when the algorithm runs n iterations without improvements. We also suggest defining n in function of l in order to simplify the parameter tunning process. Algorithm 2 presents the Stagnation Free version of LAHC. In the following experiments, we considered $n = mult \times l$, where $mult = 1,000$.

3.3.3 Avoid Side Movements - LAHC

Another way to avoid the loss of performance of LAHC in the late phase of search is to update the list only when the neighbor s' is better than the current solution i iterations before. Another modification made in this version is to leave just "less", instead of "less or equal", in the LAHC acceptance condition. It is expected that this modification has a good effect at the late phase of search, since some worse fitness will be kept in the list allowing some worsening movements. This variation will be called Avoid Side Movements LAHC, as-LAHC. The Algorithm 2 presents this variant.

3.3.4 Simulated Annealing - LAHC

Proposed by [11], the metaheuristic Simulated Annealing (SA) is a probabilistic method based on an analogy to thermodynamics, simulating the cooling of a set of heated atoms. This technique starts its search from any initial solution. The main procedure consists of a loop that randomly generates, at each iteration, one neighbor s' of the current solution s. Movements are probabilistically selected considering a temperature T and the cost variation obtained with the movement, Δ.

This algorithm was applied in the solvers of all international timetabling competition winners [13,15,4]. It also showed good results in this model of the problem, specially with larger instances. Thus, it was selected to compose a mixed approach with the LAHC algorithm. Since Simulated Annealing performance is not too affected by the initial solution fitness, we considered a mixed algorithm, where the Simulated Annealing algorithm is executed in the initial solution generating a s^* solution and then the LAHC method is executed to polish this solution, generating a final solution s^{**}. We also evaluated this mixed algorithm considering the sf-LAHC variant of LAHC.

Algorithm 2: Proposed variation of LAHC, as-LAHC

Input: Initial solution s and parameter l.
Output: Best solution s^* found.
$f_k^{'} \leftarrow f(s)\ \forall k \in \{0, ..., l - 1\}$;
$s^* \leftarrow s$;
$i \leftarrow 0$;
while $elapsedTime < timeout$ **do**
  Generate a random neighbor $s^{'} \in N(s)$;
  $v \leftarrow i \bmod l$;
  if $f(s^{'}) < f_v^{'}$ **then**
    $s \leftarrow s^{'}$;
    if $f(s) < f(s^*)$ **then**
      $s^* \leftarrow s$;
  if $f(s) \leq f_v^{'}$ **then**
    $f_v^{'} \leftarrow f(s)$;
  $i \leftarrow i + 1$;
return s^*;

The developed implementation of Simulated Annealing is described in Algorithm 3. Parameters used were $\alpha = 0,97$, $T_0 = 1$ and $SAmax = 10,000$. The method $selectMovement()$ chooses a movement according to the neighborhood probabilities previously defined.

Algorithm 3: Developed implementation of SA

Input: $f(.), N(.), \alpha, SAmax, T_0, s, timeout$
Output: Best solution s^* found.
$s^* \leftarrow s$; $IterT \leftarrow 0$; $T \leftarrow T_0$; $reheats \leftarrow 0$;
while $elapsedTime < timeout$ **do**
  while $IterT < SAmax$ **do**
    $IterT \leftarrow IterT + 1$;
    $k \leftarrow selectMovement()$;
    Generate a random neighbor $s^{'} \in N_k(s)$;
    $\Delta = f(s^{'}) - f(s))$;
    if $\Delta < 0$ **then**
      $s \leftarrow s^{'}$;
      if $f(s^{'}) < f(s^*)$ **then** $s^* \leftarrow s^{'}$;
    else
      Take $x \in [0, 1]$;
      if $x < e^{-\Delta/T}$ **then** $s \leftarrow s^{'}$;
  $T \leftarrow \alpha \times T$;
  $IterT \leftarrow 0$;
return s^*;

4 Computational Experiments

All experiments ran on an Intel ® i5 2.4 Ghz computer with 4GB of RAM computer under Ubuntu 11.10 operating system. The softwares was coded in C++ and compiled with GCC 4.6.1. The obtained results were validated by HSEval validator[1]. We considered a 1500 seconds timeout, adjusted according to the Third ITC provided benchmark, in all experiments.

The presented results are expressed by the pair x / y, where x stands for the feasibility measure and y for the quality measure. Our solver, along with our solutions and reports, can be found at GOAL-UFOP website[2]. We invite the interested reader to validate our results.

4.1 Dataset Characterization

The set of instances available from Third ITC [19] was originated from many countries and ranges from small instances to huge challenging ones. Table 1 presents the main features of considered instances.

Instance	Timeslots	Teachers	Rooms	Classes	Lessons
BrazilInstance2	25	14		6	150
BrazilInstance3	25	16		8	200
BrazilInstance4	25	23		12	300
BrazilInstance6	25	30		14	350
FinlandElementarySchool	35	22	21	291	445
FinlandSecondarySchool2	40	22	21	469	566
Aigio1stHighSchool10-11	35	37		208	532
Italy_Instance4	36	61		38	1101
KosovaInstance1	62	101		63	1912
Kottenpark2003	38	75	41	18	1203
Kottenpark2005A	37	78	42	26	1272
Kottenpark2008	40	81	11	34	1118
Kottenpark2009	38	93	53	48	1301
Woodlands2009	42	40			1353
Spanishschool	35	66	4	21	439
WesternGreeceUniversity3	35	19		6	210
WesternGreeceUniversity4	35	19		12	262
WesternGreeceUniversity5	35	18		6	184

Table 1: Features of considered instances from Third ITC

4.2 Parameter Setting

One of the key advantages of LAHC is its small number of parameters to tune. Actually, the algorithm has only one parameter, which is the length l of f'_k list. As mentioned by [2], higher values of l make the search more suitable to find better results but also more time consuming to reach them. In other hand, low values of l make the search

[1] `http://sydney.edu.au/engineering/it/` and `~jeff/hseval.cgi`

[2] Code, solutions and reports are available at `http://www.goal.ufop.br/softwares/hstt`

quick but may not reach the same solutions as in higher values. For instance, if one considers $l = 1$, the method performs exactly like the simple Hill-Climbing method.

In this sense, we ran experiments considering many values of l: l = {1, 10, 100, 500, 1000, 5000, 10000, 20000, 50000}. We choose the instances $BrazilInstance2$, $ItalyIntance4$, $SpainSchool$, $KosovaInstance$ and $NetherlandsKottenpark2009$ to determinate which value have the better performance on average. These instances were chosen since they have miscellaneous sizes and features. Tables 2, 3 and 4 present the obtained results of LAHC and its variations under the considered configurations. To perform the systematic experiments, we assumed $l = 500$, value that had good performance handling both small and large instances of the problem.

Size of l	**Brazil Instance2**	**Italy Instance4**	**Kosova Instance1**	**Netherlands Kottenpark09**	**Spain School1**
$l = 1$	1 / 67	0 / 391	1 / 114	27 / 5740	0 / 981
$l = 10$	1 / 67	0 / 388	0 / 98	28 / 7735	0 / 561
$l = 100$	1 / 46	0 / 587	0 / 40	28 / 4745	0 / 601
$l = 500$	0 / 74	0 / 82	0 / 66	28 / 4745	0 / 681
$l = 1{,}000$	0 / 149	0 / 154	0 / 54	23 / 9050	0 /1001
$l = 5{,}000$	0 / 61	0 / 63	0 / 1430	26 / 13955	0 / 460
$l = 10{,}000$	0 / 78	0 / 85	162 / 12365	22 / 7165	0 / 686
$l = 20{,}000$	0 / 58	0 / 109	429 / 22335	25 / 10375	0 / 487
$l = 50{,}000$	0 / 75	0 / 133	811 / 27396	24 / 52570	0 / 557

Table 2: Experiments considering the some values to parameter l of original LAHC

Size of l	**Brazil Instance2**	**Italy Instance4**	**Kosova Instance1**	**Netherlands Kottenpark09**	**Spain School1**
$l = 1$	1 / 67	0 / 699	1 / 113	34 / 5750	0 / 991
$l = 10$	0 / 55	0 /598	0 / 99	29 / 7610	0 /601
$l = 100$	0 / 55	1 / 792	0 / 40	29 / 10865	0 / 601
$l = 500$	0 /40	0 / 71	0 / 66	31 / 4995	0 /1027
$l = 1{,}000$	0 / 61	0 / 53	0 / 54	23 / 12110	0 / 1002
$l = 5{,}000$	0 / 61	0 / 69	0 / 1285	24 / 8940	0 /4168
$l = 10{,}000$	0 / 56	0 / 99	161 / 12420	22 / 7495	0 / 4176
$l = 20{,}000$	0 /41	0 / 128	427 / 21517	28 / 86875	0 / 4163
$l = 50{,}000$	0 / 16	0 / 9396	814 / 29172	22 / 395120	0 / 4163

Table 3: Experiments considering the some values to parameter l of sf-LAHC

4.3 Obtained Results

Table 5 presents the obtained results of LAHC method and all of its variations proposed in this work. The results of KHE engine initial solutions, ITC2012 winner approach (SA-ILS) and the standalone Simulated Annealing results are also listed in Table 5. To record the results, we considered the average result of five executions of all approaches, with random seeds. Again, in the last row, we have the ranking of all solvers considering the ITC2012 ordering procedure. The best results are highlighted in bold.

Size of l	Brazil Instance2	Italy Instance4	Kosova Instance1	Netherlands Kottenpark09	Spain School1
$l = 1$	1 / 67	0 / 400	35 / 597	30 / 5880	0 / 986
$l = 10$	1 / 67	0 / 424	36 / 1031	31 / 7960	0 / 736
$l = 100$	1 / 46	0 / 463	1 / 120	26 / 5750	0 / 1026
$l = 500$	0 / 91	0 /497	0 / 124	28 / 8060	0 / 601
$l = 1{,}000$	0 / 83	1 / 171	1 / 274	23 / 6925	0 / 744
$l = 5{,}000$	0 / 71	0 / 179	36 / 8205	22 / 8805	0 / 712
$l = 10{,}000$	0 / 123	0 / 84	188 / 12649	24 / 4920	0 /581
$l = 20{,}000$	0 / 64	0 / 87	496 / 21809	22 / 17680	0 / 726
$l = 50{,}000$	0 / 53	0 / 76	845 / 29507	23 / 116310	0 / 549

Table 4: Experiments considering the some values to parameter l of as-LAHC

4.4 Discussion of Results

As tables 2, 3 and 4 present, a poor performance was observed with LAHC if $l = 1$. This is easy to explain, as this is how Hill-Climbing performs in this problem. In general, higher sizes for f'_k list make the method and its variations able to find better solutions. In the instance *KosovaInstance1*, higher values of l compromise the performance. The refereed instance is one of the largest in the instance set, thus higher sizes of l avoid it to converge in the given timeout. In small instances, higher sizes of l presented better performance, since the local search had enough time to converge even in high sizes of l. Considering this influence on the size of l in the performance of LAHC and its variations, we set $l = 500$ to perform the systematic experiments.

To some instances, even the production of feasible solutions is already a hard task. These instances commonly define most of constraints as hard constraints. The LAHC method and its variations were able to find 12 feasible solutions out of the 18 instances in the set, one more than the ITC2012 winner. In Table 5, we can note that the LAHC algorithm and its variations were also able to outperform significantly the results of SA-ILS approach.

The standalone Simulated Annealing approach showed a below average performance. One possible explanation to this result is the high number of parameters to tune in this algorithm. Also, its parameters configuration is too much sensible to scales. For example, a parameter configuration can work nicely in a 0 / 300 initial solution but perform poorly in a 30 / 1000 initial solution. Observing the Table 5 we can easily note that the SA parameters obtained a great performance with large instances but performed poorly with the small ones. The SA-ILS algorithm was also widely outperformed by LAHC based methods.

The standalone LAHC method had a slightly better performance than the standalone SA algorithm. This is a great result, since LAHC is a new metaheuristic and has lots of possibilities of improvements. The Simulated Annealing algorithm is one of the most successfully algorithm to deal with many scheduling problems. It composed the solver of first, second and third timetabling competition winners, thus, outperforming its results is a big achievement.

The Stagnation Free version of LAHC showed surprising results, beating its original version in several instances. This could be noted specially in small instances, where sf-LAHC can keep some improvement until the timeout is reached instead of the original LAHC, which probably got stuck in these instances. Considering solutions polished by Simulated Annealing as input, the sf-LAHC also performed better than his original

Table 5: Comparsion between LAHC based approaches and SA based methods

Instance	**KHE**	**SA-ILS**	**SA**	**LAHC**	**sf-LAHC**	**SA-LAHC**	**SA-sf-LAHC**
BrazilInstance2	4 / 90	1.0 / 63.9	0.0 / 107.6	0.0 / 78.6	**0.0 / 52.8**	0.0 / 102.4	0.0 / 78.0
BrazilInstance3	3 / 240	**0.0 / 127.8**	0.0 / 170.6	0.0 / 145.6	0.0 / 137.0	0.0 / 174.0	0.0 / 160.2
BrazilInstance4	39 / 144	17.2 / 99.6	**2.0 / 167.4**	8.2 / 121.4	6.0 / 112.8	2.4 / 165.6	2.4 / 164.2
BrazilInstance6	11 / 291	4.0 / 223.5	0.0 / 332.2	1.4 / 204.8	0.6 / 166.4	0.0 / 252.6	**0.0 / 221.0**
FinlandElementarySchool	9 /30	0.0 / 4.0	0.0 / 10.0	0.0 / 3.8	**0.0 / 3.6**	0.0 / 3.8	0.0 / 3.8
FinlandSecondarySchool2	2 / 1821	0.0 / 0.4	0.0 / 1036.4	0.0 / 0.4	**0.0 / 0.2**	**0.0 / 0.2**	0.0 / 0.4
Aigio1stHighSchool10-11	14 / 757	**0.0 / 15.3**	0.0 / 397.4	0.8 / 11.4	0.8 / 11.4	0.0 / 17.8	0.0 / 20.8
Italy_Instance4	39 / 21238	0.0 / 658.4	0.0 / 13979.0	0.0 / 224.8	**0.0 / 199.4**	0.0 / 366.4	0.0 / 302.2
KosovaInstance1	1333 / 566	14.0 / 6934.4	3.0 / 5837.8	14.2 / 1504.8	**0.0 / 137.0**	4.6 / 5238.6	6.3 / 6383.8
Kottenpark2003	3 / 78440	0.6 / 90195.8	0.4 / 90052.4	1.6 / 9711.0	1.6 / 9136.8	**0.0 / 69608.2**	0.4 / 89132.2
Kottenpark2005A	35 / 23677	33.9 / 27480.4	30.0 / 33967.0	33.0 / 18671.0	32.8 / 18891.8	30.2 / 33310.4	**30.2 / 33169.6**
Kottenpark2008	63 / 140083	25.7 / 31403.7	**10.0 / 138993.8**	15.2 / 23855.0	14.4 / 23067.2	10.8 / 57476.6	10.8 / 59939.2
Kottenpark2009	55 / 211095	36.6 / 154998.5	**24.6 / 432784.0**	28.0 / 9192.0	28.8 / 8363.0	25.4 / 112948.0	25.8 / 112335.0
Woodlands2009	19 / 0	2.0 / 15.8	2.0 / 223.8	2.0 / 13.2	**2.0 / 12.0**	2.0 / 12.4	**2.0 / 12.0**
Spanish school	1 / 4103	0.0 / 865.2	0.0 / 7077.6	**0.0 / 846.0**	0.0 / 920.4	0.0 / 998.2	0.0 / 856.8
WesternGreeceUniversity3	0 / 30	0.0 / 5.6	0.0 / 28.4	**0.0 / 5.0**	**0.0 / 5.0**	0.0 / 5.4	0.0 / 5.2
WesternGreeceUniversity4	0 / 41	0.0 / 7.4	0.0 / 39.4	0.0 / 9.2	0.0 / 6.8	0.0 / 7.6	**0.0 / 6.6**
WesternGreeceUniversity5	17 / 44	**0.0 / 0.0**	0.0 / 56.0	**0.0 / 0.0**	**0.0 / 0.0**	**0.0 / 0.0**	**0.0 / 0.0**
Ranking	-	4.50	4.11	4.00	2.83	3.03	**2.53**

version. Another factor to be highlighted is the successful combination of LAHC and Simulated Annealing, which produced the best results to the problem (SA-sf-LAHC).

5 Concluding Remarks

This work presented an application of Late Acceptance Hill-Climbing algorithm and some variations to the High School Timetabling Problem model considered in the ITC2012. The LAHC algorithm showed great results. It was able to outperform the standalone Simulated Annealing approach, which is a great achievement, since the SA algorithm is widely applied to timetabling problems. It also exceeded by far the ITC2012 winner approach, a SA-ILS method. This result, coupled with another reports in literature, points that this new metaheuristic can also reach good results when applied to other problems.

We also proposed some successful variations of LAHC method. The Stagnation Free LAHC was able to beat its original version. A combination of LAHC and sf-LAHC with the Simulated Annealing was also successfully explored. The mixed SA-sf-ILS algorithm achieved the best results to this problem by now.

One great feature of LAHC algorithm is that is is very easy to be implemented. Another nice feature of LAHC is its small number of parameters to tune. It has only one parameters, while other metaheuristics generally have two or three parameters.

Some possible future works are (1) develop and evaluate another variations of LAHC as [2] suggests; (2) implement other neighborhood movements; and (3) develop a graphical user interface to allow schools and universities from all around the world to produce their instances and solve them with our solver.

Acknowledgements The authors acknowledge FAPEMIG (grants APQ-01779-10) and CNPq (grant 480388/2010-5) for supporting the development of this research.

References

1. Abuhamdah, A.: Experimental result of late acceptance randomized descent algorithm for solving course timetabling problems. In: IJCSNS- International Journal of Computer Science and Network Security, Vol. 10 No. 1, January (2010)
2. Burke, E.K., Bykov, Y.: A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems. In: PATAT '08 Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (2008). URL `http://w1.cirrelt.ca/\~{}patat2008/PATAT\_7\_PROCEEDINGS/Papers/Bykov-HC2a.pdf`
3. Burke, E.K., Bykov, Y.: The late acceptance hill-climbing heuristic. Tech. Rep. CSM-192, Department of Computing Science and Mathematics, University of Stirling (2012)
4. Fonseca, G., Santos, H., Toffolo, T., Brito, S., Souza, M.: A SA-ILS approach for the High School Timetabling Problem. In: PATAT '12 Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (2012)
5. Garey, M.R., Jonhson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco, CA, USA (1979)
6. de Haan, P., Landman, R., Post, G., Ruizenaar, H.: A case study for timetabling in a dutch secondary school. In: Lecture notes in computer science: VI Practice and theory of automated timetabling. Berlin : Springer, pp. 3867 : 267–279 (2007)
7. IDSIA: International Timetabling Competition 2002 (2012). Available at `http://www.idsia.ch/Files/ttcomp2002/`, Accessed in December / 2012
8. Kingston, J.H.: A tiling algorithm for high school timetabling. In: Lecture notes in computer science: V Practice and theory of automated timetabling. Berlin: Springer, pp. 3616 : 208–225 (2005)

9. Kingston, J.H.: Hierarchical timetable construction. In: Problems, Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (2006)
10. Kingston, J.H.: A software library for school timetabling (2012). Available at http://sydney.edu.au/engineering/it/~jeff/khe/, May 2012
11. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**, 671–680 (1983)
12. Kirkpatrick, S., Gellat, D.C., Vecchi, M.P.: Otimization by simulated annealing. In: Science, pp. 202, 671–680 (1983)
13. Kostuch, P.: The university course timetabling problem with a three-phase approach. In: Proceedings of the 5th international conference on Practice and Theory of Automated Timetabling, PATAT'04, pp. 109–125. Springer-Verlag, Berlin, Heidelberg (2005). DOI 10.1007/11593577_7. URL http://dx.doi.org/10.1007/11593577_7
14. McCollum, B.: International Timetabling Competition 2007 (2012). Available at http://www.cs.qub.ac.uk/itc2007/, Accessed in December / 2012
15. Muller, T.: ITC2007 solver description: a hybrid approach. Annals OR **172**(1), 429–446 (2009). URL http://dblp.uni-trier.de/db/journals/anor/anor172.html#Muller09
16. Nurmi, K., Kyngas, J.: A framework for school timetabling problem. In: Proceedings of the 3rd multidisciplinary international scheduling conference: theory and applications, Paris, pp. 386–393 (2007)
17. Özcan, E., Bykov, Y., Birben, M., Burke, E.K.: Examination timetabling using late acceptance hyper-heuristics. In: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, CEC'09, pp. 997–1004. IEEE Press, Piscataway, NJ, USA (2009). URL http://dl.acm.org/citation.cfm?id=1689599.1689731
18. Post, G., Ahmadi, S., Daskalaki, S., Kingston, J.H., Kyngas, J., Nurmi, C., Ranson, D.: An xml format for benchmarks in high school timetabling. In: Annals of Operations Research DOI 10.1007/s10479-010-0699-9., pp. 3867 : 267–279 (2010)
19. Post, G., Kingston, J., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., Schaerf, A.: XHSTT: an XML archive for high school timetabling problems in different countries. Annals of Operations Research pp. 1–7 (2011)
20. Santos, H.G., Uchoa, E., Ochi, L.S., Maculan, N.: Strong bounds with cut and column generation for class-teacher timetabling. Annals OR **194**(1), 399–412 (2012)
21. Valourix, C., Housos, E.: Constraint programming approach for school timetabling. In: Computers & Operations Research, pp. 30 : 1555–1572 (2003)
22. Verstichel, J., Vanden Berghe, G.: A late acceptance algorithm for the lock scheduling problem. In: S. Voss, J. Pahl, S. Schwarze (eds.) Logistik Management, Hamburg, 2-4 September 2009. Springer (2009). DOI 10.1007/978-3-7908-2362-2. URL https://lirias.kuleuven.be/handle/123456789/249443
23. Wright, M.: School timetabling using heuristic search. In: Journal of Operational Research Society, pp. 47 : 347–357 (1996)

MISTA 2013

A New Integer Linear Programming Formulation to the Examination Timetabling Problem

George H. G. Fonseca · Haroldo G. Santos

Abstract This paper presents a new integer linear programming formulation to the Examination Timetabling Problem and its experimental evaluation. The Examination Timetabling Problem consists in to assign exams to timeslots and rooms in an examination session. Students are enrolled to take these exams, so that two exams taken by the same student cannot be assigned to the same timeslot. Nowadays, heuristic approaches are the most applied technique to handle this problem. Some mathematical programming formulations were also proposed, but they never were evaluated experimentally. A new formulation to the problem was developed in this work, aiming at to be more compact and to require less memory to be handled. The obtained results were encouraging: we were able to handle some instances of the Second International Timetabling Competition. To a set of smaller instances, we were able to prove optimality in two cases and produce low cost solutions to the remaining instances. These results points that, unlike what was thought, integer programming approaches are becoming a practical approach to handle this problem.

1 Introduction

The Examination Timetabling Problem (ETTP) is faced by many educational institutions around the world. It basically consists in to assign exams to timeslots and rooms in an examination session. Students are enrolled to take these exams, so that two exams taken by the same student cannot be assigned to the same timeslot. Another constraints are considered as well, like spreading students exams.

Beyond it is practical importance, modeling the complexity of timetabling problems continues to represent an important issue in the timetabling research agenda [6]. De Werra, Asratian and Durand [13] in 2002 presented a simple model and its possible extensions for class-teacher timetabling problems.

George H. G. Fonseca
Exacts and Applied Sciences Department, Federal University of Ouro Preto
E-mail: george@decea.ufop.br

Haroldo G. Santos
Computing Department, Federal University of Ouro Preto
E-mail: haroldo@iceb.ufop.br

The ETTP was included as one of the three tracks which composed the Second International Timetabling Competition (ITC). The competition was won by Müller [9] with a Simulated Annealing [5] based approach. Many other heuristic approaches were also proposed to this problem, like Tabu Search [4,10], Genetic Algortihms [12,14], Memetic Algorithms [2] and Ant Algorithms [3,1].

This paper aims at improving the formulation proposed by [8], more specifically, to provide a more compact formulation considering its memory requirements so that it can be used in real problems. A computational study of both formulations is also presented. The computational experiments shown that our Integer Linear Programming (ILP) approach was able to handle many instances of this problem, even tough there are instances which appear to be too hard for the current generation of ILP solvers using the proposed formulations.

The remaining of this work is organized as follows: Section 2 details the ETTP as proposed by the Second International Timetabling Competition; Section 3 presents the proposed ILP formulation; Section 5 presents the computational experiments and finally, Section 6 presents some concluding remarks.

2 Examination Timetabling Problem

The problem consists of the following [7]:

- an examination session is made of a number of periods over a specified length of time; number and length of periods are provided;
- a set of exams which should be scheduled into periods;
- a set of students enrolled on individual exams; each student is enrolled on a number of exams; students enrolled on an exam are considered to "take" that examination; for each exam, the set of enrolled students is provided;
- a set of rooms with specific capacities;
- hard constraints which must be satisfied;
- soft constraints whose violation should be penalized according to pre-determined weights.

A *feasible* timetable is one in which all examinations have been assigned to a period and room and all the following hard constraints are satisfied:

- no student attends more than one examination at the same time;
- the capacity of individual rooms is not exceeded at any time throughout the examination session [1];
- period lengths are not violated;
- satisfaction of exams precedence constraints, e.g.: Exam A after Exam B;
- attendance of exams coincidence constraints, e.g.: Exam A in the same time as Exam B;
- satisfaction of exam exclusion constraints, e.g.: Exam A in a different time of Exam B;
- satisfaction of room exclusivity hard constraints, e.g.: Exam A must be the only one to use Room 101.

[1] Notice that, unlike another educational timetabling problems, exams are explicitly allowed to share rooms.

The *soft constraints* can be outlined as follows:

1. **two in a row**: no student should have two exams scheduled in consecutive periods;
2. **two in a day**: no student should have two exams in the same day;
3. **period spread**: the number of exams in a specified time period should not be greater than one for students;
4. **mixed duration**: allocation of exams with different durations in the same room and period should be avoided;
5. **front-load**: "large" exams should be be allocated in the "earlier portion" of the timetable; both "large" and "earlier portion" are user defined;
6. **undesired periods**: some periods should be avoided;
7. **undesired rooms**: some rooms should be avoided.

3 Basic Integer Programming Formulation

The basic formulation, denoted here by $\mathcal{F}$, was proposed by McCollum *et al.* [8]. It was designed to be compact and clear. McClollum *et al.* [8] also highlights that there is still room for improvement, since the encoding is not optimized towards giving the best result when used into an ILP solver. The following sections present the input data, variables, constraints and the objective function of the McColumn's [8] formulation[2].

3.1 Input Data

The following sets and parameters are directly given in the input file, or are straightforward to derive from it:

3.1.1 Exam related

E	set of exams;
$\tilde{d}_e$	duration of exam $e \in E$;
$\tilde{s}_e$	number of students who take exam $e \in E$;
A	set of pairs of exams which share students;
$\tilde{c}_{ef}$	number of students in common for exams $(e, f) \in A$;
$\tilde{F}$	subset of exams $\tilde{F} \subseteq E$ which are subjected to front-load constraints;
D	set of durations which an exam can have;
u_{ed}	a binary constant which is 1 if exam e has duration $d \in D$, 0 otherwise.

3.1.2 Student related

S	set of students;
T_s	set of exams that a student $s \in S$ must attend.

3.1.3 Room related

R	set of rooms;
$\acute{c}_r$	capacity of room $r \in R$;
$\acute{w}_r$	a weight that specifies the penalty for using room $r \in R$.

[2] Some notation modifications were made in order to clarify the presentation of the ILP model.

3.1.4 Period related

P	set of periods;
$\hat{d}_p$	duration of period $p \in P$;
$\hat{F}$	subset of timeslots $\hat{F} \subseteq P$ which are subjected to front-load constraints;
$\hat{s}_{pq}$	a binary constant that is 1 if periods p and q occurs on the same day, 0 otherwise;
g	minimal "gap" between two exams for a student;

3.1.5 Constraint related

B	set of pairs of exams (e, f) in which f should be scheduled after e;
C	set of pairs of exams (e, f) in which e should lie in the same period as f;
G	set of pairs of exams (e, f) in which e cannot lie in the same period as f;
H	set of exams $e \in H$ which should be scheduled to an exclusive room;
w_i	weight for violating the i-th constraint presented in Section 2; additional indices may be used when the penalty refers to some specific allocation, e.g.: $w_{6,p}$ indicates the penalty weight for violating the "undesired periods" soft constraint for period p;

3.2 Variables

The main decision variables are related to the assignment of an exam to a period and room:

$$x_{ep} = \begin{cases} 1 & \text{if exam } e \text{ is assigned to period } p \\ 0 & \text{otherwise} \end{cases}$$

$$y_{er} = \begin{cases} 1 & \text{if exam } e \text{ is assigned to room } r \\ 0 & \text{otherwise} \end{cases}$$

The auxiliary variables are those which have their values induced by the assignment of the main variables. They are used to write the constraints and to compute the objective function.

To encode the room capacity constraint, the following variables were added:

$$z_{epr} = \begin{cases} 1 & \text{if exam } e \text{ is assigned to period } p \text{ and room } r \\ 0 & \text{otherwise} \end{cases}$$

with the meaning that

$$z_{epr} \equiv x_{ep} \times y_{er} \quad \forall e \in E, \forall p \in P, \forall r \in R \tag{1}$$

these variables must be linked using ILP constraints:

$$x_{ep} = \sum_{r \in R} z_{epr} \quad \forall e \in E, \forall p \in P \tag{2}$$

$$y_{er} = \sum_{p \in P} z_{epr} \quad \forall e \in E, \forall r \in R \tag{3}$$

To evaluate the usage of a room at each period considering exams of different durations we use the following additional binary variables:

$$u_{dpr} = \begin{cases} 1 & \text{if some exam of duration } d \text{ is assigned to room } r \text{ and period } p \\ 0 & \text{otherwise} \end{cases}$$

For measuring violations in the various soft constraints previously described we employ integer variables $v_i \in \mathbb{Z}^+$, such that v_i indicates how much the i-th soft constraint was violated. Additional indexes may be used if the violation refers to a specific pair of exams, e.g.: v_{1ef} indicates the violation for constraint 1 (two in a row) considering exams e and f.

3.3 Constraints

Assuming the given sets, parameters and variables, the constraints can be modeled as follows:

3.3.1 Hard constraints

$$\sum_{p \in P} x_{ep} = 1 \quad \forall e \in E \tag{4}$$

$$\sum_{r \in R} y_{er} = 1 \quad \forall e \in E \tag{5}$$

$$\sum_{r \in R} y_{er} = \sum_{p \in P} z_{epr} \quad \forall e \in E \tag{6}$$

$$\sum_{p \in P} x_{ep} = \sum_{r \in R} z_{epr} \quad \forall e \in E \tag{7}$$

$$\sum_{e \in E} \tilde{s}_e z_{epr} \leq c_r \quad \forall p \in P, r \in R \tag{8}$$

$$\tilde{d}_e x_{ep} \leq \hat{d}_p \quad \forall p \in P, e \in E \tag{9}$$

$$\sum_{e \in T_s} x_{ep} \leq 1 \quad \forall s \in S, p \in P \tag{10}$$

$$x_{fp_1} + x_{ep_2} \leq 1 \quad \forall (e,f) \in B, p_1 \in P, p_2 \in P : p_1 \leq p_2 \tag{11}$$

$$x_{ep} = x_{fp} \quad \forall (e,f) \in C, p \in P \tag{12}$$

$$x_{ep} + x_{fp} \leq 1 \quad \forall (e,f) \in G, p \in P \tag{13}$$

$$x_{ep} + x_{fp} + y_{er} + y_{fr} \leq 3 \quad \forall e \in H, f \in E : e \neq f, \forall p \in P, r \in R \tag{14}$$

Constraints 4 and 5 ensure the allocation of a timeslot and a room for each exam, respectively. Constraints 7 and 6 link z variables with the main decision variables. Constraints 9 limit the number of students per room. Constraints 10 avoid exam conflicts for students. Constraints 11 impose precedence constraints among events. Constraint 12 and 13 force the allocations of two events in the same period or in different periods, respectively. Finally, constraints 11 forbid the allocation of additional events in the same timeslot and room where a event which requires an exclusive room is alocated.

3.3.2 Soft constraints

$$x_{ep_1} + x_{fp_2} \leq 1 + v_{1ef} \quad \forall (e,f) \in A, p_1, p_2 \in P : |p_1 - p_2| = 1 \wedge \hat{s}_{p_1,p_2} = 1 \quad (15)$$

$$x_{ep_1} + x_{fp_2} \leq 1 + v_{2ef} \quad \forall (e,f) \in A, p_1, p2_{\in} P : |p_1 - p_2| \geq 2 \wedge \hat{s}_{p_1 p_2} = 1b \quad (16)$$

$$x_{ep_1} + x_{ep_2} \leq 1 + v_{3ef} \quad \forall (e,f) \in A, p_1, p_2 \in P : 1 \leq |p_1 - p_2| \leq g \quad (17)$$

$$x_{ep} + y_{er} \leq 1 + u_{dpr} \quad \forall e \in E, p \in P, r \in R : \hat{d}_p = d \wedge \hat{d}_e = d \quad (18)$$

$$\sum_{d \in D} u_{dpr} \leq 1 + v_{4pr} \quad \forall p \in P, r \in R \quad (19)$$

Some soft constraint can be evaluated directly in decision variables, specifically the front-load constraints and the undesired period/room constraints. Constraints 15, 16 and 17 are related to pairs of events which share one or more students: the proximity of these events is checked for consecutive periods, days and periods which are close. Constraints 18 force auxiliary variables u referring to different durations to be activated whenever an event of this duration is allocated in a given period and room. These variables are used in constraints 19 to compute violations of the avoid mixed durations soft constraint.

3.3.3 Objective function

The objective function includes auxiliary variables v and weight constants w which indicate the importance of each soft constraint:

$$\sum_{(e,f) \in A} \sum_{i=1}^{3} \tilde{c}_{ef} w_i v_{ief} + \sum_{p \in P} w_{6p} x_{ep} + \sum_{r \in R} w_{7r} y_{er} + \sum_{p \in P} \sum_{r \in R} w_4 v_{4pr} + \sum_{e \in \tilde{F}} \sum_{p \in \hat{F}} w_5 x_{ep}$$

4 New Formulation

The main drawback observed in the previous formulation is the large amount of required memory to instantiate real problems (see Section 5). Our proposed formulation, denoted here by $\mathcal{F}^{'}$ is more compact, needing needs less memory to be handled by Integer Programming solvers since it has a much sparser constraint matrix. In this formulation we modified some constraints of the basic formulation and added new sets and variables. The added elements and the new formulation will be outlined in the following subsections.

4.1 New Variables and Sets

In order to make easier to specify some constraints of the problem, the following sets were added to the formulation.

D : set of days;

$\grave{P}_d$: periods belonging to day $d \in D$.

Unlike the previous formulation, whose decision variables were all binary, this formulation introduces general integer variables:

$$a_e \in \mathbb{Z}^+ : \text{the allocation period for exam } e$$

as additional binary variables we have:

$$q_{ed} = \begin{cases} 1 & \text{if exam } e \text{ is assigned to some timeslot in day } d \\ 0 & \text{otherwise} \end{cases}$$

the new set of variables is linked with the original ones by means of the following constraints:

$$a_e = \sum_{p \in P} p x_{ep} \quad \forall e \in E \tag{20}$$

$$q_{ed} = \sum_{p \in \grave{P}_d} x_{ep} \quad \forall e \in E, d \in D \tag{21}$$

Before proceeding to describe the new constraints we discuss the additions recently made: the number of variables is increased by $|E|+|E|\times|D|$.The number of constraints is increased in the same number. The impact in decreasing the number of constraints and non-zeros, as we will show, is significantly larger.

We now define hard constraints (11, 12), respectively, as:

$$a_e \geq a_f + 1 \quad \forall (e,f) \in B \tag{22}$$

$$a_e = a_f \quad \forall (e,f) \in C \tag{23}$$

thus, requiring $O(|E|^2)$ constraints instead of $O(|E|^2 \times |P|^2)$ to express precedence and coincidence constraints.

Soft constraints 16 and 17 can now be replaced by:

$$v_{2ef} \geq q_{ed} + q_{fd} - 1 \quad \forall (e,f) \in A, d \in D \tag{24}$$

$$v_{3ef} \geq x_{ep} + \sum_{p' \in P : p' \geq p \wedge p' \leq p+g-1} x_{fp'} - 1 \quad \forall (e,f) \in A, p \in P \tag{25}$$

The number of required constraints for checking for violations of the two-in-a-day soft constraint is reduced from $O(|E|^2 \times |P|^2)$ to $O(|E|^2 \times |D|)$, while in the period-spread soft constraint we reduce from $O(|E|^2 \times |P|^2)$ to $O(|E|^2 \times |P|)$. These later constraints are reduced, however, at the expense of more dense constraints.

5 Computational Experiments

All experiments ran on an Intel ® i5 2.4 Ghz computer with 4GB of RAM under Ubuntu 11.10 operating system. The integer programming solver used was CPLEX 12.5 ®. We considered an one hour timeout to each instance and formulation. Our integer programming model along with our solutions and reports can be found at `https://sites.google.com/site/georgehgfonseca/producaoacademica/examTT.rar`. We invite the interested reader to validate our results.

5.1 Dataset Characterization

We considered two sets of instances used to evaluate this formulation. The first was used on the Second International Timetabling Competition and is available at `http://www. cs.qub.ac.uk/itc2007/examtrack`. They reproduce real problems originated from many countries and are representative of a large range of different systems [8]. Table 1 presents the main features of this set of instances. The column density presents the percentage chance of two exams have students in common.

Instance	Exams	Students	Periods	Rooms	Density
Exam_1	607	7891	54	7	5.05
Exam_2	870	12743	40	49	1.17
Exam_3	934	16439	36	48	2.62
Exam_4	273	5045	21	1	15.00
Exam_5	1018	9253	42	3	0.87
Exam_6	242	7909	16	8	6.16
Exam_7	1096	14676	80	15	1.93
Exam_8	598	7718	80	8	4.55

Table 1: Features of Second ITC instances

The second set of instances was originated from the work of [11]. It consists in a real dataset from eight semesters in three consecutive years of Yeditepe University examination session. Table 2 presents the main features of this set of instances.

Instance	Exams	Students	Periods	Rooms	Density
yue20011	126	559	18	2	0.18
yue20012	141	591	18	2	0.18
yue20013	26	234	6	2	0.25
yue20021	162	826	21	2	0.18
yue20022	182	869	21	2	0.17
yue20023	38	420	6	1	0.20
yue20031	174	1125	18	2	0.15
yue20032	210	1185	18	2	0.14

Table 2: Features of Yeditepe University instances

5.2 Results and Discussion

First we present the dimensions of the generated constraint matrix for the linear programs: Table 5.2 includes the number of columns, rows and non-zeros for each formulation. As it can be seen, the number of non-zeros for the larger instances is significantly reduced using our formulation.

In our experiments we observed that both formulations are weak in terms of lower bounds, i.e.: all lower bounds at the root node relaxation were zero. This was expected because our major goal was to provide a compact formulation. Stronger formulations would require, for example, the addition of cuts which would make the linear programs even larger.

instance	$\mathcal{F}$			$\mathcal{F}'$		
	columns	rows	non-zeros	columns	rows	non-zeros
exam_1	300,385	6,363,450	19,295,800	318,592	2,171,275	11,044,329
exam_2	1,815,496	6,402,489	17,492,091	1,827,673	6,042,492	16,661,359
exam_4	28,488	798,468	2,521,667	30,669	366,345	1,609,707
exam_5	188,337	2,957,777	8,986,963	203,604	1,212,632	5,462,130
exam_6	43,324	684,215	1,936,163	45,499	284,330	1,196,776
exam_7	1,464,908	23,437,305	69,035,857	1,509,841	7,115,999	36,736,063
exam_8	468,667	20,069,959	60,402,461	493,182	3,140,765	26,826,608
yue20011	11,322	76,557	255,180	12,201	77,436	243,426
yue20012	13,347	93,273	306,777	14,331	94,257	290,877
yue20013	799	3,583	8,637	874	3,684	8,670
yue20021	17,577	139,011	475,384	18,870	140,304	450,439
yue20022	20,260	162,256	539,712	21,713	163,709	509,243
yue20023	944	5,263	12,320	1,055	5,374	12,280
yue20031	16,788	126,243	429,096	18,003	127,458	408,138
yue20032	20,973	157,941	521,175	22,440	159,408	492,972

Table 3: Dimensions of the linear programs

instance	$\mathcal{F}$	$\mathcal{F}'$
exam_1	OM	113,625
exam_2	OM	OM
exam_3	OM	OM
exam_4	46,647	68,483
exam_5	111,424	118,466
exam_6	48,920	40,755
exam_7	OM	OM
exam_8	OM	OM
yue20011	101	124
yue20012	162	283
yue20013	**29**	**29**
yue20021	80	158
yue20022	921	874
yue20023	**46**	**46**
yue20031	803	438
yue20032	1.364	1.353

Table 4: Costs of the best solutions found by CPLEX 12.5 within an 1 hour timeout

Table 5.2 presents the obtained results of both formulations applied to the Yeditepe University and to the Second ITC instances. Cells with "OM" indicate executions were the solver was aborted due to an out of memory error. The smaller memory requirements of our proposed formulation allowed it to be executed in one more instance without exceeding the available memory. For two instances, yue20013 and yue20023 the obtained solution had its optimality proved.

6 Concluding Remarks

We presented a new formulation to the Examination Timetabling Problem and its experimental evaluation. McCollum *et al.* [8] mentioned that his formulation was

usable only in instances smaller than the ones given in the Second ITC. Nevertheless, nowadays one similar formulation was able to handle some instances of Second ITC.

To a set of smaller instances of the problem, the integer programming approach achieved great results. It found and proved the optimality in two instances, besides got good solutions to the remaining ones.

The obtained results pointed out that the integer programming approach is suitable to small instances of this problem. Moreover, integer programming models are easier to implement and to modify than heuristic approaches, specially considering the addition of new constraints. Thus this approach may be useful in practical situations where the user wants to solve small instances of the problem and/or adapt the model to attend his needs.

Some possible future works are: (1) develop improved encodings to this problem as McCollum *et al.* [8] suggests and (2) tune integer programming solver configurations to deal with specific features of this problem.

Acknowledgements The authors acknowledge FAPEMIG (grants APQ-01779-10) and CNPq (grant 480388/2010-5) for supporting the development of this research.

References

1. Azimi, Z.N.: Hybrid heuristics for examination timetabling problem. Applied Mathematics and Computation pp. 705–733 (2005)
2. Burke, E.K., Newall, J.P., Weare, R.F.: A memetic algorithm for university exam timetabling. In: Selected papers from the First International Conference on Practice and Theory of Automated Timetabling, pp. 241–250. Springer-Verlag, London, UK, UK (1996). URL `http://dl.acm.org/citation.cfm?id=646429.692755`
3. Dowsland, K.A., Thompson, J.M.: Ant colony optimization for the examination scheduling problem. Journal of the Operational Research Society **56**(4), 426–438 (2005). DOI doi:10.1057/palgrave.jors.2601830
4. Kendall, G., Hussin, N.M.: An investigation of a tabu search based hyper-heuristic for examination timetabling. In: PP 309328. RBAIET AlHeuristic, Meta-Heuristic and Hiper-Heuristic Approaches 11 Kotzan J and Evanson R, pp. 465–469. Springer (2005)
5. Kirkpatrick, S., Gellat, D.C., Vecchi, M.P.: Otimization by simulated annealing. In: Science, pp. 202, 671–680 (1983)
6. McCollum, B.: University timetabling: Bridging the gap between research and practice. In: in Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, pp. 15–35. Springer (2006)
7. McCollum, B., McCullam, P., Burke, E.K., Parkes, A.J., Qu, R.: The Second International Timetabling Competition: Examination Timetabling Track. QUB/IEEE/Tech/ITC2007/Exam/v4.0/17 (2007)
8. McCollum, B., McMullan, P., Parkes, A., Burke, E., Qu, R.: A new model for automated examination timetabling. Annals of Operations Research **194**(1), 291–315 (2012)
9. Müller, T.: ITC2007 solver description: a hybrid approach. Annals OR **172**(1), 429–446 (2009). URL `http://dblp.uni-trier.de/db/journals/anor/anor172.html#Muller09`
10. Paquete, L., Stutzle, T.: T.: Empirical analysis of tabu search for the lexicographic optimization of the examination timetabling problem. In: In: 4th International Conference on the Practice and Theory of Automated Timetabling, pp. 21–23. Springer-Verlag (2002)
11. Parkes, A.J., Ozcan, E.: Properties of yeditepe examination timetabling benchmark instances. In: 8th International Conference on Practice and Theory of Automated Timetabling. Springer-Verlag, London, UK, UK (2010)
12. Ross, P., Marin-Blazquez, J., Hart, E.: Hyper-heuristics applied to class and exam timetabling problems. In: Evolutionary Computation, 2004. CEC2004. Congress on, vol. 2, pp. 1691 – 1698 Vol.2 (2004). DOI 10.1109/CEC.2004.1331099
13. de Werra, D., Asratian, A.S., Durand, S.: Complexity of some special types of timetabling problems. Journal of Scheduling **5**(2), 171–183 (2002). DOI 10.1002/jos.97. URL `http://dx.doi.org/10.1002/jos.97`

14. Wong, T., Ct, P., Gely, P.: Final exam timetabling: A practical approach. In: Canadian Conference on Electrical and Computer Engineering, pp. 12–15 (2002)

MISTA 2013

Line search method for solving a non-preemptive strictly periodic scheduling problem

Clément Pira · Christian Artigues

Abstract We study a non-preemptive strictly periodic scheduling problem. This problem arises for example in the avionic field where a set of N periodic tasks (measure of a sensor, data presentation, etc.) has to be scheduled on P processors distributed on the plane. In this article, we consider an existing heuristic which is based on the notion of equilibrium. Following a game theory analogy, each task tries successively to optimize its own schedule and therefore to produce the best response, given the other schedules. We present a new method to compute the best response, and a propagation mechanism for non-overlapping constraints, which significantly improves this heuristic. These improvements allow the heuristic to compare favorably with MILP solutions.

1 Periodic scheduling problem and its MILP formulation

1.1 Context of the work

The problem of scheduling a set of N periodic tasks on a set of P processors has a long history. The preemptive case has received more attention, starting with the work of Liu and Layland [11]. Under this hypothesis, good schedulability conditions can be derived, and the problem is efficiently solved by the EDF algorithm (Earliest Deadline First), at least in the uniprocessor context. In the non-preemptive case, schedulability conditions are generally much weaker and the problem becomes NP-complete in the strong sense, even with one processor, as shown by Jeffay *et al* [8]. The latter still consider a loose notion of periodicity in which the tasks have to execute during each time period, but not always at the same relative position. In order to enforce regularity of the schedule, some works have considered the problem of minimizing the jitter [4]. In the extreme case where the jitter is imposed to be zero, the problem is qualified

Clément Pira
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse, France
E-mail: clement.pira@laas.fr

Christian Artigues
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse, France
E-mail: christian.artigues@laas.fr

as a strictly periodic problem. In the avionic field, a preemptive or loosely periodic schedule is problematic because of the need to certify solutions. In this article, we consider a non-preemptive strictly periodic scheduling problem [9,6,7,1]. As shown by Baruah *et al* [3] and Korst *et al* [9,10], this is a strongly NP-complete problem. In this problem, each task i has a fixed period T_i and a processing time p_i. They are subject to non-overlapping constraints : no two tasks assigned to the same processor can overlap during any time period (see Figure 1). A solution is given by an assignment of the tasks to the processors and, for each task by the start time t_i (offset) of one of its occurrences, the other starting at $t_i + kT_i$ by strict periodicity.

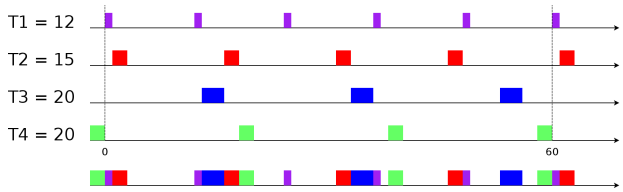


Fig. 1 $N = 4$ non-overlapping periodic tasks on $P = 1$ processor

In the following, unlike the papers mentioned above, we adopt a more general model in which processing times p_i are generalized by positive latency delays $l_{i,j} \geq 0$ (or time lags). The former case is the particular case where $l_{i,j} = p_i$ for all other tasks j. This variant has been studied mostly in the monoperiodic case [5,14]. The proposed heuristic adapts easily to this generalization.

In this paper, we only consider the case where the offsets t_i are integers. This hypothesis is important to prove convergence of the method described in section 2.1.2. Since the problem is periodic, all the periods T_i also need to be integers[1]. However the latency delays need not be integer *a priori*, especially in the case of the optimization problem presented in section 1.2.2.

1.2 Definition of the uniprocessor problem

1.2.1 Non-overlapping constraints

In the formulation of a uniprocessor periodic scheduling problem, a latency delay $l_{i,j} \geq 0$ has to be respected whenever an occurrence of a task j starts after an occurrence of a task i. Said differently, the smallest positive difference between two such occurrences has to be greater than $l_{i,j}$. Using Bézout identity, the set $(t_j+T_j\mathbb{Z})-(t_i+T_i\mathbb{Z})$ of all the possible differences is equal to $(t_j - t_i) + g_{i,j}\mathbb{Z}$ where $g_{i,j} = \gcd(T_i, T_j)$. The smallest positive representative of this set is $(t_j - t_i) \mod g_{i,j}$ (in particular, we consider a classic positive modulo, and not a signed modulo). Therefore, we simply consider the following constraint :

$$(t_j - t_i) \bmod g_{i,j} \geq l_{i,j}, \qquad \forall (i,j),\ i \neq j \tag{1}$$

[1] Indeed, if t_i is an integer solution then $t_i + T_i$ should also be an integer solution, hence their difference too, *i.e.* T_i.

Note that except for the modulo, these constraints have a clear analogy with classical precedence constraints of the form $t_j - t_i \geq l_{i,j}$. However, they are much harder to handle since they induce an exclusion between tasks. In particular, a disjunctive constraint $t_j - t_i \geq l_{i,j} \vee t_i - t_j \geq l_{i,j}$ can always be represented by the two constraints $(t_j - t_i) \bmod T \geq l_{i,j}$ and $(t_i - t_j) \bmod T \geq l_{j,i}$ for a large enough period T.

Classically, processing times are strictly positive, however our model allows zero delays. Since equation (1) is trivially satisfied when $l_{i,j} = 0$, we introduce an additional graph $\mathcal{G}$ containing the arcs (i, j) for which $l_{i,j} > 0$, and we consider equation (1) only for those couples. As we will see, the constraints associated with (i, j) and (j, i) work naturally together and some complications are introduced when only one of the couples has a strictly positive delay. To avoid special cases, we will suppose in the following that the graph $\mathcal{G}$ is symetric[2] : two delays $l_{i,j}$ and $l_{j,i}$ are either both zero, or both strictly positive. With this hypothesis, a consequence of proposition 1 (see Appendix) is that equation (1) can be replaced by the following interval constraint for each couple $(i, j) \in \mathcal{G}$ with $i < j$ (see [9]) :

$$l_{i,j} \leq (t_j - t_i) \bmod g_{i,j} \leq g_{i,j} - l_{j,i}, \qquad \forall (i,j) \in \mathcal{G},\ i < j \tag{2}$$

Since $\mathcal{G}$ is symetric, it can be seen as an undirected graph. We will write $\mathcal{G}(i)$ for the set of neighbors of i, *i.e.* the set of tasks with which i is linked through non-overlapping constraints.

1.2.2 Objective to maximize

Robustness is concerned with the problem of computing solutions which can withstand uncertainties on the parameters. In the context of hard real time system, like the one encountered in the avionic field, the uncertainties on the delays (processing times, latencies) are particularly critical. A task may last longer than expected, and therefore these parameters are generally overestimated *a priori*. Instead of this, we define here a measure of the robustness of a schedule which capture its capacity to handle increases of the durations. For this, we suppose that the possible variations of the delays are proportional to their original values : tasks with large processing times are more likely to be subject to large overrun. Hence, all the delays $l_{i,j}$ are made proportional to a common factor $\alpha \geq 0$ that we try to optimize (see [1,2]).

$$\max \quad \alpha \tag{3}$$

$$s.t. \quad l_{i,j}\alpha \leq (t_j - t_i) \bmod g_{i,j} \leq g_{i,j} - l_{j,i}\alpha \qquad \forall (i,j) \in \mathcal{G},\ i < j \tag{4}$$

$$t_i \in \mathbb{Z} \qquad \forall i \tag{5}$$

$$\alpha \geq 0 \tag{6}$$

Whenever a solution with $\alpha \geq 1$ is found, we have a solution for the feasibility problem. If a larger factor is found, the schedule can handle critical situations where all the delays have been multiplied by this factor.

The addition of this objective is especially usefull for heuristic approaches (e.g. local search), for which we need a criterium to select better solutions than the current one. The α-criterium given above amounts to maximize space between tasks, and this idea is at the base of the heuristic presented in this paper (see section 2).

[2] Note that this hypothesis is quite natural (and can be enforced anyway) : if $l_{i,j} > 0$, then the tasks i and j are constrained not to start at the same time. Hence the same is true for the tasks j and i. Therefore, $l_{j,i}$ can also be supposed strictly positive.

1.3 MILP formulation of the multiprocessor problem

In this section, we present a MILP formulation of the multiprocessor problem which will be used to compare the performance of our heuristic.

1.3.1 MILP formulation

In the multiprocessor case, the scheduling problem is coupled with an assignment problem. Thus, we introduce binary variables $a_{i,k}$ which indicate if a task i is assigned to a processor k, and variables $x_{i,j}$ which indicate if i and j are on different processors. These variables must satisfy (8) and (9). For a given couple of tasks $(i,j) \in \mathcal{G}$, with $i<j$, the non-overlapping constraint (4) can be rewritten as $l_{i,j}\alpha \leq t_j - t_i + g_{i,j}q_{i,j} \leq g_{i,j} - l_{j,i}\alpha$. Indeed, the modulo $(t_j - t_i) \bmod g_{i,j}$ is the only element $t_j - t_i + g_{i,j}q_{i,j}$ in the interval $[0, g_{i,j})$, where $q_{i,j}$ is an additional 'quotient variable'. More precisely, the non-overlapping constraints have to be satisfied whenever two tasks are on the same processor. Therefore, we replace the term $l_{i,j}\alpha$ by $l_{i,j}\alpha - l_{i,j}\alpha_{\max}x_{i,j}$, where $\alpha_{\max}$ is an upper bound on the value of α (see section 1.3.3). When i and j are on the same processor ($x_{i,j}=0$), we find back the original term $l_{i,j}\alpha$, otherwise we obtain a negative term $l_{i,j}\alpha - l_{i,j}\alpha_{\max}$ which makes the constraint trivially satisfiable. This yields constraints (10) and (11).

$$\max \quad \alpha \tag{7}$$

$$\sum_k a_{i,k} = 1 \qquad \forall i \tag{8}$$

$$x_{i,j} \leq 2 - a_{i,k} - a_{j,k} \qquad \forall k, \quad \forall (i,j) \in \mathcal{G}, i<j \tag{9}$$

$$t_j - t_i + g_{i,j}q_{i,j} \geq l_{i,j}\alpha - \alpha_{\max}l_{i,j}x_{i,j} \qquad \forall (i,j) \in \mathcal{G}, i<j \tag{10}$$

$$t_j - t_i + g_{i,j}q_{i,j} \leq g_{i,j} - l_{j,i}\alpha + \alpha_{\max}l_{j,i}x_{i,j} \qquad \forall (i,j) \in \mathcal{G}, i<j \tag{11}$$

$$t_i \in \mathbb{Z} \qquad \forall i \tag{12}$$

$$q_{i,j} \in \mathbb{Z} \qquad \forall (i,j) \in \mathcal{G}, i<j \tag{13}$$

$$x_{i,j} \in [0,1] \qquad \forall (i,j) \in \mathcal{G}, i<j \tag{14}$$

$$a_{i,k} \in \{0,1\} \qquad \forall k, \forall i \tag{15}$$

1.3.2 Bounds on the offsets and quotients

The previous system is invariant under translation $t_i \mapsto t_i + n_i T_i$. More precisely, if (t_i) is feasible and if (n_i) is a vector of integers, then the solution (t'_i) defined by $t'_i = t_i + n_i T_i$ is also feasible, provided we define new quotients $q'_{i,j} = q_{i,j} + n_i \frac{T_i}{g_{i,j}} - n_j \frac{T_j}{g_{i,j}}$. In the previous reasoning, the only important fact is that $q'_{i,j}$ is again integer since $g_{i,j}$ divides T_i and T_j. Therefore, it remains true if we replace T_i with an updated period :

$$T_i^* = \operatorname{lcm}((g_{i,j})_{j \in \mathcal{G}(i)}) = \gcd(T_i, \operatorname{lcm}((T_j)_{j \in \mathcal{G}(i)})) \tag{16}$$

The updated period T_i^* always divides T_i and we can have a strictly smaller value for example if a prime factor p only occurs in one period T_i (or more generally if a factor p occurs with multiplicity m in T_i but with strictly smaller multiplicities in any other periods of a task connected to i). Indeed, even if the (T_i) are the initial parameters,

only their GCD appear in the constraints. Therefore, if a prime factor occurs in only one period, then it completely disappear in the $g_{i,j}$. By replacing the initial periods (T_i) by the updated ones (T_i^*), we simply remove some irrelevant factors. In the following, we will suppose that the periods have been updated to have no proper factors, *i.e.* $T_i = T_i^*$.

Since $t_i \bmod T_i = t_i + n_i T_i$ for some $n_i \in \mathbb{Z}$, we deduce that $(t_i \bmod T_i)$ is also a solution. We can therefore impose additionnal bounds $t_i \in [0, T_i - 1]$. And since, we have $0 \leq t_j - t_i + g_{i,j} q_{i,j} \leq g_{i,j}$, these bounds on the offsets immediately induce associated bounds on the quotients :

$$1 - \frac{T_j}{g_{i,j}} \leq q_{i,j} \leq \frac{T_i}{g_{i,j}} \tag{17}$$

In particular, if T_j divides T_i, then $q_{i,j} \geq 0$ (in fact $q_{i,j} \in [0, T_i/T_j]$), which shows that in the harmonic case, we can impose positive variables. Conversely if T_i divides T_j then $q_{i,j} \leq 1$. Finally, if $T_i = T_j$, then $q_{i,j} \in \{0, 1\}$ which shows that the monoperiodic case correspond to the case of binary variables.

1.3.3 Upper bound on the α-value

Let i and j be two tasks on the same processor. Then, constraint (4) implies the following upper bound : $\alpha \leq g_{i,j}/(l_{i,j} + l_{j,i})$. Taking the integrality assumption into account, we get an even tighter upper bound :

$$\alpha \leq \alpha_{\max}^{i,j} = \max\left(\frac{1}{l_{i,j}} \left\lfloor \frac{g_{i,j}}{l_{i,j} + l_{j,i}} l_{i,j} \right\rfloor \ , \ \frac{1}{l_{j,i}} \left\lfloor \frac{g_{i,j}}{l_{i,j} + l_{j,i}} l_{j,i} \right\rfloor \right) \tag{18}$$

These bounds are illustrated on Figure 2. In the monoprocessor case, we deduce that $\alpha_{\max} = \min_{i,j} \alpha_{\max}^{i,j}$ is an upper bound on the value of α. In the multiprocessor case, we have at least the trivial upper bound $\alpha_{\max} = \max_{i,j} \alpha_{\max}^{i,j}$. However, since $\alpha_{\max}$ is used as a 'big-M' in the MILP formulation, we would like the lowest possible value in order to improve the efficiency of the model. For this, we solve a preliminary model, dealing only with the assignment (hence without variables (t_i) and $(q_{i,j})$), in which non-overlapping constraints (10-11) are replaced by the following weaker constraint :

$$\alpha \leq \alpha_{\max}^{i,j} + (\alpha_{\max} - \alpha_{\max}^{i,j}) x_{i,j}, \qquad \forall (i,j) \in \mathcal{G},\ i < j \tag{19}$$

Intuitively, this constraint indicates that α should be less than $\alpha_{\max}^{i,j}$ if i and j are assigned to the same processor ($x_{i,j} = 0$), otherwise it is bounded by $\alpha_{\max}$ which is the trivial upper bound. Solving this model is much faster than for the original one (less that $1s$ for instances with 4 processors and 20 tasks). It gives us a new value $\alpha_{\max}$ which can be used in the original model.

2 An equilibrium-based heuristic

Given a set of precedence constraints, the Bellman-Ford algorithm allows to find compatible offsets. A way to implement this algorithm is to cyclically choose a task and to push it forward, up to the first feasible offset. In the case of non-overlapping constraints, we can find the next feasible offset for a task, if it exists, by a method analogous to the propagation mechanism presented in section 2.2.5. However there isn't necessarily

a solution and a similar mechanism which would cyclically move one task at a time to the next feasible location would have only a small chance to converge towards a feasible solution. The addition of the α-value solves this problem : instead of choosing the next feasible location (possibly nonexistent), we search for the best location with respect to this additional objective α, what we call the best-response method. By moving the tasks round after round, we increase the space between them. This is essentially the principle of the heuristic presented in this section.

2.1 Heuristic and the mutiprocessor best response method

2.1.1 The multiprocessor best response method

The main component of the algorithm is called the best response procedure. It takes its name from a game theory analogy. Each task is seen as an agent which tries to optimize its own assignment a_i and offset t_i, while the other assignments (a_j^*) and offsets (t_j^*) are fixed. Instead of using a binary vector $(a_{i,k})$ as in the MILP formulation, we represent an assignment more compactly by a variable $a_i \in [1, P]$. For each agent i, and each processor p, we first define a method BestOffset_i^p which returns the best possible offset for task i on processor p. More formally, the BestOffset_i^p procedure consists in solving the following program :

$$(BO_i^p) \quad \max \quad \alpha \tag{20}$$

$$s.t. \quad l_{j,i}\alpha \leq (t_i - t_j^*) \bmod g_{i,j} \leq g_{i,j} - l_{i,j}\alpha \qquad \begin{smallmatrix}\forall j \in \mathcal{G}(i) \\ a_j^* = p\end{smallmatrix} \tag{21}$$

$$t_i \in \mathbb{Z} \tag{22}$$

$$\alpha \geq 0 \tag{23}$$

Contraints (21) are the non-overlapping contraints of task i with other tasks j currently on this processor ($a_j^* = p$). We define N_p to be the number of such contraints (we have $\sum_p N_p = |\mathcal{G}(i)| \leq N - 1$). Note that there are only two variables, t_i and α, since the other offsets and assignments $(t_j^*, a_j^*)_{j \in \mathcal{G}(i)}$ are parameters. Following the discussion in section 1.3.2, the previous system is invariant under tranlation by T_i, and even by the possibly smaller value $T_i^p = \text{lcm}((g_{i,j})_{j \in \mathcal{G}(i) | a_j^* = p})$. Hence, we can impose t_i to belong to $[0, T_i^p - 1]$. Since t_i is integer, we can trivially solve this program by computing the α-value for each of the offsets $\{0, \cdots, T_i^p - 1\}$, using the following expression, and selecting the best one.

$$\alpha = \min_{\substack{j \in \mathcal{G}(i) \\ a_j^* = p}} \min \left(\frac{(t_i - t_j^*) \bmod g_{i,j}}{l_{j,i}} \ , \ \frac{(t_j^* - t_i) \bmod g_{i,j}}{l_{i,j}} \right) \tag{24}$$

A much better approach to solve this program will be presented in section 2.2. Given this procedure, we define a method $\text{MultiProcBestResponse}_i$ which returns the best assignment and offset for the task i, given the current assignments and offsets (t_j^*, a_j^*) of all the tasks. In order to choose the assignment, an agent simply compute the best-offset on each processor and select the best one. It starts with the current processor a_i, which has priority in case of equality. Some of the next processors can sometimes be skipped. Indeed, for a given processor p, we can compute an upper bound on the objective : $\alpha_{\max}^p = \min_{j \in \mathcal{G}(i) | a_j^* = p} \alpha_{\max}^{i,j}$ (similar to section 1.3.3). Therefore, if the current best solution found on previous processors is already better than this upper

bound, there is no way to improve the current solution with processor p, which can be skipped. The multiprocessor best-response procedure is summarized in Algorithm 1.

Algorithm 1 The multiprocessor best-response

procedure MULTIPROCBESTRESPONSE$_i$($(t_j^*)_{j\in I}, (a_j^*)_{j\in I}$)
  $a_i \leftarrow a_i^*$ ▷ The initial best assignment is the current one
  $(t_i, \alpha) \leftarrow \text{BESTOFFSET}_i^{a_i^*}((t_j^*)_{j\in I}, (a_j^*)_{j\in I})$ ▷ The current best offset and value
  for all $p \neq a_i^*$ **do** ▷ We test the other processors
    $\alpha_{\max}^p \leftarrow \min_{\substack{j\in\mathcal{G}(i)\\ a_j^*=p}} \alpha_{\max}^{i,j}$ ▷ We compute an upper bound on α when i is on p
    if $\alpha_{\max}^p > \alpha$ **then** ▷ An improvement can possibly be found on p
      $(\tau, \beta) \leftarrow \text{BESTOFFSET}_i^p((t_j^*)_{j\in I}, (a_j^*)_{j\in I})$ ▷ We search the best offset on p
      if $\beta > \alpha$ **then** ▷ An improvement has been found on p
        $a_i \leftarrow p$; $t_i \leftarrow \tau$; $\alpha \leftarrow \beta$ ▷ We update the best solution
      end if
    end if
  end for
  return (t_i, a_i, α)
end procedure

Since the BESTOFFSET_i^p method runs in $O(T_i N_p)$, the MULTIPROCBESTRESPONSE$_i$ method runs in $O(T_i N)$.

2.1.2 The concept of equilibrium and principle of the method to find one

Definition 1 A solution (t_i, a_i) is an equilibrium *iff* no task i can improve its assignment or offset using procedure MULTIPROCBESTRESPONSE$_i$.

In order to find an equilibrium, the heuristic uses a counter N_{stab} to count the number of tasks known to be stable, *i.e.* which cannot be improved. It starts with an initial solution (for example randomly generated) and tries to improves this solution by a succession of unilateral optimizations. On each round, we choose cyclically a task i and try to optimize its schedule, *i.e.* we apply MULTIPROCBESTREPONSE$_i$. If no improvement was found, then one more task is stable, otherwise we update the assignment and offset of task i and reinitialize the counter of stable tasks. We continue until N tasks are stable. This is summarized in Algorithm 2.

Remark 1 The main reason for the use of integers is that it allows to guarantee the convergence of the heuristic. The termination proof relies on the fact that the underlying game is an ordinal potential game. This potential strictly increases on each round, preventing the algorithm from cycling. And since it can take only a finite number of values, the algorithm converges after a finite number of steps (we refer to [1] for the proof of termination and correction). This is not the case with fractional offsets, for which the heuristic is unlikely to converge in a finite number of steps.

An equilibrium is only an approximate notion of optimum. Hence, in order to find a real optimum, the idea is now to run the previous heuristic several times with different randomly generated initial solutions, and to keep the best result, following a standard multistart scheme.

Algorithm 2 The heuristic

```
1: procedure ImproveSolution((t_j)_{j∈I}, (a_j)_{j∈I})
2:     N_stab ← 0                                   ▷ The number of stabilized tasks
3:     i ← 0                                        ▷ The task currently optimized
4:     while N_stab < N do                          ▷ We run until all the tasks are stable
5:         (new_t_i, new_a_i, α_i) ← MultiProcBestResponse_i((t_j)_{j∈I})
6:         if new_t_i ≠ t_i or new_a_i ≠ a_i then
7:             t_i ← new_t_i; a_i ← new_a_i         ▷ We have a strict improvement for task i
8:             N_stab ← 1                           ▷ We restart counting the stabilized tasks
9:             α ← α_i
10:        else                                     ▷ We do not have a strict improvement
11:            N_stab ← N_stab + 1                  ▷ One more task is stable
12:            α ← min(α_i, α)
13:        end if
14:        i ← (i + 1) mod N                        ▷ We consider the next task
15:    end while
16:    return (α, (t_j)_{j∈I}, (a_i)_{j∈I})
17: end procedure
```

2.2 The best-offset procedure on a given processor

We now need to implement the BestOffset_i^p method, *i.e.* to solve (BO_i^p), more efficiently than by the trivial linear procedure presented in section 2.1.1. Since the latter runs in $O(T_i N_p)$, any method should at least be faster. In [2], the authors propose a method consisting in computing the α-value only for a set of precomputed intersection points (in the next section we will see that a fractional optimum is at the intersection of an increasing and a decreasing line). This already improves the trivial procedure. In the following, we present an even more efficient method which relies on the fact that (BO_i^p) is locally a two-dimensional linear program.

2.2.1 Structure of the solution set

If we try to draw the function $\alpha = (t_i - t_j^*) \bmod g_{i,j}/l_{j,i}$ representing the optimal value of α respecting one constraint $(t_i - t_j^*) \bmod g_{i,j} \geq l_{j,i}\alpha$ when t_i takes rational values, we obtain a curve which is piecewise increasing and discontinuous since it becomes zero on $t_j^* + g_{i,j}\mathbb{Z}$. In the same way, if we draw $\alpha = (t_j^* - t_i) \bmod g_{i,j}/l_{i,j}$, we obtain a curve which is piecewise decreasing and becomes zero at the same points. It is therefore more natural to consider the two constraints jointly, *i.e.* (4), which gives a continuous curve (see Figure 2) [3].

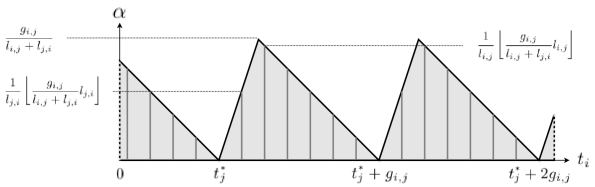


Fig. 2 Possible values for (t_i, α) when constrained by a single task j

[3] Note that if $\mathcal{G}$ was not symetric, we would need to consider some degenerate cases were one of the slope (increasing or decreasing) would be infinite (since $l_{i,j} = 0$ or $l_{j,i} = 0$).

Given some fixed offsets $(t_j^*)_{j\in\mathcal{G}(i)}$, the set of solutions (t_i, α) of (BO_i^p) is the intersection of the sets decribed above for each $j \in \mathcal{G}(i)$, $a_j^* = p$ (see Figure 3). It is composed of several adjacent polyhedra. We can give an upper bound on the number n_{poly} of such polyhedra. A polyhedron starts and ends at zero points. For a given constraint j, there is $T_i/g_{i,j}$ zero points in $[0, T_i - 1]$, hence n_{poly} is bounded by $T_i \sum_{\substack{j\in\mathcal{G}(i)\\ a_j^*=p}} \frac{1}{g_{i,j}}$. This upper bound can be reached when the offsets are fractional, since in this case, we can always choose the offsets t_j^* such that the sets of zero points $(t_j^* + g_{i,j}\mathbb{Z})_{j\in\mathcal{G}(i)}$ are all disjoint. In the case of integer offsets, there is obvioulsy at most T_i zero points in the interval $[0, T_i - 1]$ and therefore, at most T_i polyhedra.

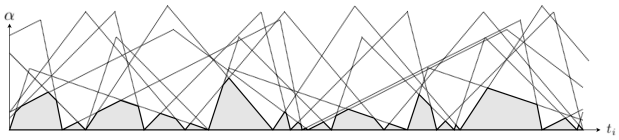


Fig. 3 Possible values for (t_i, α) constrained by all tasks $j \in \mathcal{G}(i)$, $a_j^* = p$

When solving the best-response method, we try to find the new location t_i which maximizes the associated α-value.

2.2.2 Local (two dimensional) polyhedron

We want to compute the local polyhedron which contains a reference offset t_i^{ref}. Locally, the constraint $(t_i - t_j^*) \bmod g_{i,j} \geq l_{j,i}\alpha$ is linear, of the form $t_i - o_j \geq l_{j,i}\alpha$. Here, o_j is the largest $\tau \leq t_i^{\text{ref}}$ such that $(\tau - t_j^*) \bmod g_{i,j} = 0$. In the same way, we can compute the decreasing constraint $o'_j - t_i \geq l_{i,j}\alpha$. In this case o'_j is the smallest $\tau \geq t_i^{\text{ref}}$ such that $(o'_j - \tau) \bmod g_{i,j} = 0$. By proposition 2 (see Appendix), we have :

$$o_j = t_i^{\text{ref}} - (t_i^{\text{ref}} - t_j^*) \bmod g_{i,j} \qquad \text{and} \qquad o'_j = t_i^{\text{ref}} + (t_j^* - t_i^{\text{ref}}) \bmod g_{i,j} \tag{25}$$

Therefore, we obtain a local polyhedron (see figure 4). Note that when $(t_i^{\text{ref}} - t_j^*) \bmod g_{i,j} > 0$, we simply have $o'_j = o_j + g_{i,j}$ by proposition 1 (see Appendix). However, when $(t_i^{\text{ref}} - t_j^*) \bmod g_{i,j} = 0$, we have $o_j = o'_j = t_i^{\text{ref}}$. In this case, the polyhedron is degenerated since it contains only $\{t_i^{\text{ref}}\}$, and the α-value at t_i^{ref} is zero. Instead of computing this polyhedron, we prefer to choose either the polyhedron on the right, or on the left (see figure 5). If we choose the one on the right, this amounts to defining o'_j to be the smallest $\tau > t_i^{\text{ref}}$ which sets the constraint to zero. By proposition 2, we have $o'_j = t_i^{\text{ref}} + g_{i,j} - (t_i^{\text{ref}} - t_j^*) \bmod g_{i,j}$. Therefore, this simply amounts to enforcing $o'_j = o_j + g_{i,j}$. Choosing the polyhedron on the left amounts to defining o'_j using (25) and to enforcing $o_j = o'_j - g_{i,j}$.

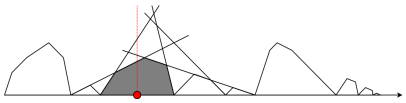


Fig. 4 Selection of the polyhedron containing a reference offset t_i^{ref}

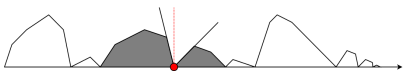


Fig. 5 Two possibilities in the degenerate case $\alpha = 0$

2.2.3 Solving the local best offset problem

Once the local polyhedron has been defined, the problem is now to solve the following MILP :

$$
\begin{array}{lllll}
(Loc{-}BO_i^p) & \max & \alpha & & (26)\\
& s.t. & t_i - l_{j,i}\alpha \geq o_j & \forall j \in \mathcal{G}(i),\ a_j^* = p & (27)\\
& & t_i + l_{i,j}\alpha \leq o'_j & \forall j \in \mathcal{G}(i),\ a_j^* = p & (28)\\
& & t_i \in \mathbb{Z} & & (29)
\end{array}
$$

We can first search for a fractional solution, and for this we can use any available method of linear programming. However, since the problem is a particular two dimensional program, we can give special implementations of these methods. In the following, we present a simple primal simplex approach, which runs in $O(N_p^2)$ in the worst case but has a good behaviour in practice. A local polyhedron is delimited by increasing and decreasing lines, and the fractional optimum is at the intersection of two such lines. Hence a natural way to find the optimum is to try all the possible intersections between an increasing line, of the form $t_i - l_{j,i}\alpha = o_j$, and a decreasing line, of the form $t_i + l_{i,k}\alpha = o'_k$, and to select the one with the smallest α-value. The coordinates of these intersection points are given by[4] :

$$t_i = \frac{l_{j,i}o'_k + l_{i,k}o_j}{l_{j,i} + l_{i,k}} \qquad \text{and} \qquad \alpha = \frac{o'_k - o_j}{l_{j,i} + l_{i,k}} \tag{30}$$

Since there is N_p lines of each kinds, the algorithm runs in $O(N_p^2)$. In practice, a better approach (with the same worst case complexity) is to start with a couple of increasing and decreasing lines, and alternatively to try to improve the decreasing line (see Figure 6), then the increasing one (see Figure 7), and so on, until no improvement is made. The overall solving process is illustrated on Figure 8.

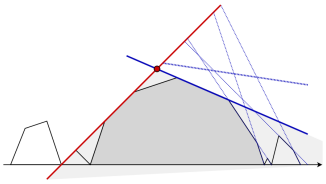


Fig. 6 The lowest intersection point of a fixed increasing line with decreasing lines

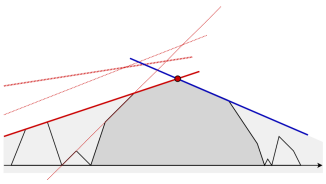


Fig. 7 The lowest intersection point of a fixed decreasing line with increasing lines

Remark 2 Suppose we just improved the decreasing line (Figure 6) and we get a new intersection point. We know that the whole local polyhedron lies inside the cone oriented below and defined by the increasing and the decreasing line. Then all the decreasing lines with a smaller slope than the new decreasing line, *i.e.* with a larger delay $l_{i,j}$, lie completely outside this cone, and therefore cannot be active at the optimum. We can therefore drop these lines in the subsequent phases. In other words, the delay $l_{i,j}$ of the decreasing line strictly decreases with time (except for the first improvement, since the initial decreasing line can be any line). The same is true for the increasing line.

[4] Note that in the computation only the α-coordinate is needed since the t_i-coordinate of the selected intersection point can be computed afterward by $t_i = o_j + l_{j,i}\alpha$

Let's define a phase to be the computation of a new increasing or decreasing line. Then the previous remark implies that the number of "decreasing" phases (*resp.* "increasing" phases) is bounded by the number of distinct delays $l_{i,j}$ (*resp.* $l_{j,i}$). Since a phase runs in $O(N_p)$, the method runs in $O(N_p^2)$ in the worst case of distinct delays. This gives us a method with the same complexity to compute an integral solution since once a fractional solution has been found, we can deduce an integral solution with an additional computation in $O(N_p)$. Indeed, if t_i is integer, then (t_i, α) is the desired solution. Otherwise we can compute the α-values α_- and α_+ associated with $\lfloor t_i \rfloor$ and $\lceil t_i \rceil$ and take the largest one. Note that since $\lfloor t_i \rfloor$ (*resp.* $\lceil t_i \rceil$) is on the increasing phase (*resp.* decreasing phase), only the corresponding constraints are needed to compute α_- (*resp.* α_+) :

$$\alpha_- = \min_{\substack{j \in \mathcal{G}(i) \\ a_j^* = p}} (\lfloor t_i \rfloor - o_j)/l_{j,i} \qquad \text{and} \qquad \alpha_+ = \min_{\substack{k \in \mathcal{G}(i) \\ a_k^* = p}} (o'_k - \lceil t_i \rceil)/l_{i,k} \tag{31}$$

We call this a primal approach since this is essentially the application of the primal simplex algorithm. However the primal simplex is not applied on $(Loc{-}BO_i^p)$ but on its dual. A dual approach is illustrated on Figure 9 and presented in a technical report [13]. From a theoretical perspective, this approach is outperformed by Megiddo algorithm [12] which allows to find a solution in $O(N_p)$. However Megiddo algorithm is more complex and generally slower in practice.

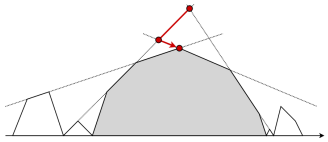


Fig. 8 Finding the fractional optimum with a primal simplex approach

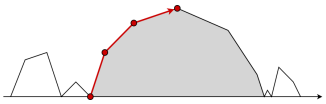


Fig. 9 Finding the fractional optimum with a dual simplex approach

2.2.4 The case of processing times

In the case of processing times, no such refinement as Megiddo algorithm is needed in order to obtain a complexity in $O(N_p)$. In fact, in this case the primal simplex method already runs in $O(N_p)$ since it stops after three phases. Indeed, in the case of processing times, we have $l_{i,j} = p_i$, which means than all the decreasing line have the same slope $-1/p_i$ (see Figure 10). Thus, even if initially there are N_p decreasing lines with equations $o'_j - t_i = \alpha p_i$, the one with the smallest o'_j will be selected after the first phase. In a second phase, we search for a better increasing line. The third phase can be dropped since we will not find a better decreasing line : we are at the optimum.

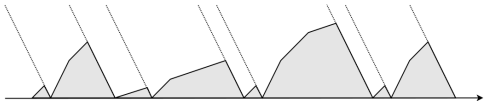


Fig. 10 All the decreasing lines are parallel in the case of processing times

2.2.5 Finding the next improving offset

After a first local optimization, we obtain a solution t_i^{lb} with value α_{lb}, which becomes a lower bound. In the rest of the procedure, we are only interested by polyhedra which could improve this value. As α_{lb} is improved, more and more polyhedra will lie completely below this level and will be skipped (see darker polyhedra in Figure 11).

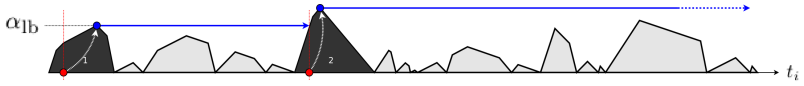


Fig. 11 Propagating up to the next strictly improving offset

Starting from a current solution t_i^{lb} with value α_{lb}, we want to find a new integer solution t_i greater than the current solution which strictly improves its value. Hence, this solution should satisfy :

$$\alpha_{\text{lb}} l_{j,i} < (t_i - t_j^*) \bmod g_{i,j} < g_{i,j} - \alpha_{\text{lb}} l_{i,j} \qquad \begin{matrix} \forall j \in \mathcal{G}(i) \\ a_j^* = p \end{matrix} \tag{32}$$

However, since the middle expression gives an integer value, we can round the bounds and obtain the equivalent interval constraint (34). Moreover, among all the possible solutions, we would like to find the smallest one :

$$\min \quad t_i \tag{33}$$

$$s.t. \quad \lfloor \alpha_{\text{lb}} l_{j,i} \rfloor + 1 \leq (t_i - t_j^*) \bmod g_{i,j} \leq g_{i,j} - \lfloor \alpha_{\text{lb}} l_{i,j} \rfloor - 1 \qquad \begin{matrix} \forall j \in \mathcal{G}(i) \\ a_j^* = p \end{matrix} \tag{34}$$

$$t_i \in \{t_i^{\text{lb}}, \cdots, t_i^{\text{end}}\} \tag{35}$$

Here, t_i^{end} (which will be defined in the next section) indicates when to stop the search (since there is possibly no strictly improving solution). In order to solve this program, we will start with $t_i = t_i^{\text{lb}}$ and propagate on the right until we find a feasible solution. On a given iteration, we choose a constraint j and check if it is satisfied. We could check the two inequalities of the interval constraint (34), however we can also remark that this is equivalent to :

$$(t_i - t_j^* - \lfloor \alpha_{\text{lb}} l_{j,i} \rfloor - 1) \bmod g_{i,j} \leq g_{i,j} - \lfloor \alpha_{\text{lb}} l_{i,j} \rfloor - \lfloor \alpha_{\text{lb}} l_{j,i} \rfloor - 2 \tag{36}$$

Therefore we can compute $r = g_{i,j} - (t_i - t_j^* - \lfloor \alpha_{\text{lb}} l_{j,i} \rfloor - 1) \bmod g_{i,j}$. If $r < \lfloor \alpha_{\text{lb}} l_{i,j} \rfloor + \lfloor \alpha_{\text{lb}} l_{j,i} \rfloor + 2$, the constraint is violated. In this case, we compute the smallest offset τ strictly greater than the current one, and which satisfies the current constraint, *i.e.* we compute $\tau > t_i$ such that $(\tau - t_j^* - \lfloor \alpha_{\text{lb}} l_{j,i} \rfloor - 1) \bmod g_{i,j} = 0$. By proposition 2, we have $\tau = t_i + r$. For this offset τ, the current constraint is now verified. We set $t_i = \tau$ and continue the process with the next constraint. We cycle along the constraints, until no update is made during N_p successive rounds or the offset t_i becomes greater than t_i^{end}. In the former case, all the constraints are satisfied by the current offset, otherwise there is no solution and we return the special value $\varnothing$. This procedure is summarized in algorithm 3. For convenience, we suppose here that the constraints $j \in \mathcal{G}(i),\ a_j^* = p$ are numbered from 0 to $N_p - 1$.

Algorithm 3 A propagation procedure to find an improving integral solution

procedure FINDIMPROVINGINTEGRALSOLUTION($\alpha_{\text{lb}}, t_i^{\text{lb}}, t_i^{\text{end}}$)
  $t_i \leftarrow t_i^{\text{lb}}$ ▷ The current offset
  $N_{\text{sat}} \leftarrow 0$ ▷ The number of satisfied constraints
  $j \leftarrow 0$ ▷ The current constraint evaluated
  while $N_{\text{sat}} < N_p$ **do**
    $r \leftarrow g_{i,j} - (t_i - t_j^* - \lfloor \alpha_{\text{lb}} l_{j,i} \rfloor - 1) \bmod g_{i,j}$
    if $r \geq \lfloor l_{i,j}\alpha_{\text{lb}} \rfloor + \lfloor l_{j,i}\alpha_{\text{lb}} \rfloor + 2$ **then**
      $N_{\text{sat}} \leftarrow N_{\text{sat}} + 1$ ▷ One more constraint is satisfied
    else
      $t_i \leftarrow t_i + r$ ▷ Otherwise, we go to the first feasible offset
      if $t_i > t_i^{\text{end}}$ **then return** $\varnothing$ ▷ We reach the end without solution
      $N_{\text{sat}} \leftarrow 1$ ▷ We restart counting the satisfied constraint
    end if
    $j \leftarrow (j+1) \bmod N_p$ ▷ We consider the next constraint
  end while
  return t_i ▷ We return an improving offset
end procedure

2.2.6 Solving the best response problem

We are now able to describe a procedure which solves (BO_i^p). We saw that t_i can be supposed to belong to $[0, T_i^p - 1]$. More generally we can start at any initial offset t_i^{start}, for example the current value of t_i, and run on the right until we reach the offset $t_i^{\text{end}} = t_i^{\text{start}} + T_i^p - 1$. We can compute the local polyhedron (on the right) which contains the current offset. Using the primal simplex method, we solve the associated problem $(Loc{-}BO_i^p)$. We obtain a new local optimum $(t_i^{\text{lb}}, \alpha_{\text{lb}})$ which becomes our lower bound. We then use the propagation procedure to reach the next improving solution. We are in a new polyhedron and we restart the local optimization at this point, which gives us a better value. We continue until the propagation mechanism reaches t_i^{end} (see Figure 11). In the end, we obtain a best offset $t_i \in [t_i^{\text{start}}, t_i^{\text{end}}]$. If needed, we can consider $t_i \bmod T_i^p$ which is an equivalent solution in $[0, T_i^p - 1]$.

If we use Megiddo algorithm to solve the local problems, or if we use the primal simplex approach in the case of processing times, this procedure runs in $O(N_p n_{\text{poly}})$, which is bounded by $O(N_p T_i^p)$. Indeed, one iteration of the propagation mechanism runs in $O(1)$. Since the propagation progresses of one polyhedron every N_p iterations[5], the end of the period is reached after at most $N_p n_{\text{poly}}$ iterations. During this process, there is at most n_{poly} additionnal local optimizations each one running in $O(N_p)$. Note that if we use a primal simplex approach, the local optimizations run in $O(N_p^2)$ in the worst case, but most of the polyhedra are skipped by the propagation mechanism which still runs in $O(N_p n_{\text{poly}})$.

[5] Let us define a cycle to be N_p consecutive iterations of the propagation mechanism and let us show that after 2 cycles, the current offset t_i doesn't lie in the same local polyhedron. Note that after a cycle, the algorithm either stops or an update has been made. Consider a constraint j updated during the second cycle. If j was not updated during the first cycle, then this contraint was satisfied during the first cycle (hence the current offset was in a given polyhedron of Figure 2), but violated during the second. Therefore, at the end of second cycle, the offset has been pushed to the next polyhedron of Figure 2. If j was already updated during the first cycle, then the current offset has been pushed forward to the next feasible solution two times for this constraint. Therefore, the local polyhedron has also changed.

3 Results

We have tested the method on non-harmonic instances generated using the procedure described in [7] : the periods were choosen in the set $\{2^x 3^y 50 \mid x \in [0,4], y \in [0,3]\}$ and the processing times were generated following an inverse exponential distribution and averaging at about 20% of the period of the task. The experiments were performed on an Intel Core i5-480M 2.66GHz with 4GB of memory.

Table 1 presents the results on 15 instances with $N = 20$ tasks and $P = 4$ processors. Columns 4-8 contain the results of our new version of the heuristic. The value $\mathbf{start}_{10s}$ represents the number of time the heuristic was launched with different initial solutions during 10s. From this value, we compute $\mathbf{time}_{single} = 10/\mathbf{start}_{10s}$ which represents the average time needed for one execution of the heuristic. The value $\mathbf{start}_{sol}$ represents the number of starts needed to obtain the best result. Hence, the quantity $\mathbf{time}_{sol} = \mathbf{start}_{sol} \cdot \mathbf{time}_{single}$ gives approximately the time needed to obtain this best result (C++ timers were not precise enough so we preferred this approximation). Column $\mathbf{time}_{sol}$ of the MILP formulation (columns 2-3) represents the time needed by the Gurobi solver to obtain the best solution during a period of 200s[6]. In addition to being much faster, the heuristic sometimes obtains better results (see instances 1 and 3). This table also includes the results of the original heuristic presented in [1,2] (columns 9-10). The field $\mathbf{time}_{single}$ represents the time needed for a single run of their version of the heuristic. Averaging at 1.95s, these values are compatible with the results presented in [2]. The field $\mathbf{gain}_{speed}$, which is the ratio of the two values $\mathbf{time}_{single}$, shows that our version of the heuristic is incomparably faster (about 3200 times on these instances).

	MILP (200s)		New heuristic (10s)					Original heuristic [2]	
id	α_{MILP}	$\mathbf{time}_{sol}$	$\alpha_{heuristic}$	$\mathbf{starts}_{10s}$	$\mathbf{time}_{single}$	$\mathbf{starts}_{sol}$	$\mathbf{time}_{sol}$	$\mathbf{time}_{single}$	$\mathbf{gain}_{speed}$
0	2.5	159	2.5	15062	$6.64e^{-4}$	30	0.01992	1.43	2154
1	2	18	2.01091	15262	$6.55e^{-4}$	15	0.00983	3.27	4991
2	1.6	6	1.6	11018	$9.08e^{-4}$	2	0.00182	1.52	1675
3	1.6	4	1.64324	11970	$8.35e^{-4}$	45	0.03759	4.34	5195
4	2	5	2	10748	$9.30e^{-4}$	1	0.00093	3.48	3740
5*	3	7	3	20428	$4.90e^{-4}$	1	0.00049	1.63	3330
6	2.5	54	2.5	20664	$4.84e^{-4}$	30	0.01452	1.44	2976
7	2	19	2	14040	$7.12e^{-4}$	1	0.00071	0.23	323
8	2.12222	8	2.12222	17365	$5.76e^{-4}$	3	0.00173	1.03	1789
9*	2	11	2	26304	$3.80e^{-4}$	3	0.00114	2.42	6366
10	1.12	6	1.12	28778	$3.47e^{-4}$	69	0.02398	0.72	2072
11	2.81098	20	2.81098	13355	$7.49e^{-4}$	7	0.00524	3.78	5048
12	1.5	7	1.5	11444	$8.74e^{-4}$	4	0.00350	0.27	309
13	1.56833	49	1.56833	25997	$3.85e^{-4}$	1	0.00038	1.77	4601
14	2	8	2	21606	$4.63e^{-4}$	2	0.00093	1.85	3997

Table 1 Results of the MILP, the heuristic of [2], and the new version of the heuristic on instances with $P = 4$ processors and $N = 20$ tasks

These good results have encouraged us to perform additional tests on large instances (50 processors, 1000 tasks). Table 2 presents the results for 10 instances, where $\mathbf{starts}_{1000s}$ is the number of runs performed during 1000s, $\mathbf{time}_{single}$ is the average

[6] We fix a timeout of 200s because the solver almost never stop even after 1h of CPU time (except for instances 5 and 9 for which optimality has been proved in $7s$ and $20s$ respectively).

time for one run, $\mathbf{starts}_{\text{sol}}$ is the round during which the best solution was found, and $\mathbf{time}_{\text{sol}}$ is the corresponding time. This shows that our heuristic can give feasible solutions ($\alpha \geq 1$) in about 1min, while these instances cannot even be loaded by the MILP solver (since they involve millions of variables and constraints). In order to evaluate the contribution of the propagation mechanism to the solving process, we also present results where the propagation has been replaced by a simpler mechanism : once we are at a local optimum, we follow the decreasing line active at this point, until we reach the x-axis; this gives us a next reference offset and therefore a next polyhedron. While the impact of the propagation is quite small in the case of small instances, we see on these large instances that the propagation mechanism accelerates the process by a factor of 37 (average ratio of the two $\mathbf{time}_{\text{single}}$ values).

	With propagation					Without propagation				
id	$\alpha_{\text{heuristic}}$	$\mathbf{starts}_{1000s}$	$\mathbf{time}_{\text{single}}$	$\mathbf{starts}_{\text{sol}}$	$\mathbf{time}_{\text{sol}}$	$\alpha_{\text{heuristic}}$	$\mathbf{starts}_{1000s}$	$\mathbf{time}_{\text{single}}$	$\mathbf{starts}_{\text{sol}}$	$\mathbf{time}_{\text{sol}}$
0	1	386	2.59	6	13.2	1	16	66.07	6	405.45
1	1.16452	200	5.01	35	197.6	1.05	6	177.9	5	885.06
2	1.1	111	9.04	78	694.17	1.04	4	261.19	1	405.32
3	1.21795	86	11.69	8	62.57	1.176	4	300.03	1	157.18
4	1	105	9.57	1	7.24	1	4	309.70	1	306.34
5	1.66555	121	8.30	114	952.62	1.58795	2	798.01	1	800.95
6	1	240	4.17	6	26.49	1	15	72.29	6	403.05
7	1.2	88	11.54	37	484.91	1	4	235.44	1	278.17
8	1.39733	83	12.09	47	468.93	1.21127	2	618.75	2	1237.5
9	1.2	77	13.29	1	20.58	1.2	2	487.03	1	420.58

Table 2 Results of the new version of the heuristic on instances with $P = 50$ processors and $N = 1000$ tasks, with $timeout = 1000s$

4 Conclusion

In this paper, we have proposed an enhanced version of a heuristic, first presented in [1, 2], and allowing to solve a NP-hard strictly periodic scheduling problem. More specifically, we present an efficient way to solve the best-response problem. This procedure alternates between local optimizations and an efficient propagation mechanism which allows to skip most of the polyhedra. The results show that the new heuristic greatly improves the original one and compares favorably with MILP solutions. In particular, it can handle instances out of reach of the MILP formulation.

A Appendix

Proposition 1 *If* $x \bmod a = 0$*, then* $(-x) \bmod a = 0$*, otherwise* $(-x) \bmod a = a - x \bmod a$*. In particular* $x \bmod a > 0 \Leftrightarrow (-x) \bmod a > 0$*.*

Proposition 2 *(1) The smallest* $y \geq a$ *such that* $(y - b) \bmod c = 0$ *is given by* $y = a + (b - a) \bmod c$*. (2) The smallest* $y > a$ *such that* $(y - b) \bmod c = 0$ *is given by* $y = a + c - (a - b) \bmod c$*. (3) The largest* $y \leq a$ *such that* $(y - b) \bmod c = 0$ *is given by* $y = a - (a - b) \bmod c$*. (4) The largest* $y < a$ *such that* $(y - b) \bmod c = 0$ *is given by* $y = a - c + (b - a) \bmod c$*.*

Acknowledgements This work was funded by the French Midi-Pyrenee region (allocation de recherche post-doctorant n°11050523) and the LAAS-CNRS OSEC project (Scheduling in Critical Embedded Systems).

References

1. A. Al Sheikh. Resource allocation in hard real-time avionic systems - Scheduling and routing problems. PhD thesis, LAAS, Toulouse, France, 2011.
2. A. Al Sheikh, O. Brun, P.E. Hladik, B. Prabhu. Strictly periodic scheduling in IMA-based architectures. Real Time Systems, Vol 48, N°4, pp. 359-386, 2012.
3. S. Baruah, L. Rousier, I. Tulchinsky, D. Varvel. The complexity of periodic maintenance. In: Proceedings of the International Computer Symposium, 1990.
4. S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari. Scheduling periodic task systems to minimize output jitter. In International Conference on Real-Time Computing Systems and Applications, pp. 62-69, Hong Kong, 1999.
5. W. Dauscha, H.D. Modrow, A. Neumann. On Cyclic Sequence Type for Constructing Cyclic Schedules. Zeitschrift für Operations Research, Vol. 29, N°1, pp. 1-30, 1985.
6. F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, A. Wiese. Scheduling periodic tasks in a hard real-time environment. In: Automata, languages and programming, pp. 299-311, 2010.
7. F. Eisenbrand, K. Kesavan, R.S. Mattikalli, M. Niemeier, A.W. Nordsieck, M. Skutella, J. Verschae, A. Wiese. Solving an Avionics Real-Time Scheduling Problem by Advanced IP-Methods. ESA 2010, pp. 11-22, 2010.
8. K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of period and sporadic tasks. In: Proceedings of the 12^{th} IEEE Symposium on Real-Time Systems, pp. 129-139, 1991.
9. J. Korst, E. Aarts, J.K. Lenstra, J. Wessels. Periodic multiprocessor scheduling. In: E.H.L. Aarts, M. Rem, J. van Leeuwen (eds.) PARLE 1991. LNCS, Vol. 505, pp. 166-178. Springer, Heidelberg, 1991.
10. J. Korst. Periodic multiprocessors scheduling. PhD thesis, Eindhoven university of technology, Eindhoven, the Netherlands, 1992.
11. C.L. Liu, J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM, Vol. 20, N°1, pp. 46-61, 1973.
12. N. Megiddo. Linear-Time Algorithms for Linear Programming in R^3 and Related Problems. SIAM J. Comput., Vol. 12, N°4, pp.759-776, 1983.
13. C. Pira and C. Artigues. An efficient best response heuristic for a non-preemptive strictly periodic scheduling problem. Technical Report 12668 LAAS-CNRS, Toulouse, 2012.
14. P. Serafini, W. Ukovich. A Mathematical Model for Periodic Scheduling Problems. SIAM J. Disc. MATH, Vol. 2, N°4, pp. 550-581, 1989.

MISTA 2013

A GIS-Based Emergency Response System for Disaster Relief Operations

Onur Demir • Linet Özdamar

Abstract In this work, we present a Geographical Information System (GIS) based emergency response system for efficient collaboration and timely planning of disaster relief operations for delivery of vital materials and evacuation of injured people. Our approach is based on a centralized GIS server application where material delivery and injured pickup requests are organized and transmitted to a Helicopter Operations Scheduling System (HOPS) that generates routes for the available helicopter pool. The HOPS response system uses a hierarchical optimization approach where the relief network is clustered and a routing problem is solved on an aggregated network first. Then, after passing some parameters to the cluster routing problems, these are solved in parallel. The GIS based response system is illustrated on a post-earthquake scenario.

1 Introduction

Delivery of vital supplies such as food, water or medicine and evacuating injured people are two important tasks in an emergency response operation. In disasters where the land transportation infrastructure is heavily damaged, the best way to do this task is by using helicopters. In major earthquakes and floods, injured people can safely and quickly be evacuated by helicopters. We assume that the available relief material supplies are limited and stored at different warehouses outside the disaster zone. These have to be picked up by helicopters and dropped off at designated locations. Noting that the available emergency patient capacities at local hospitals are also limited, helicopters are also assigned to the evacuation task of transporting needy people to hospitals with free service capacity. If the emergency area is large, such operations require a large number of helicopters that have to be coordinated efficiently. Due to the magnitude of demands with regard to vehicle capacity split deliveries and pickups are allowed. The number of available vehicles is limited. A vehicle can start and end its tour at different locations. After a major disaster, managing the delivery and pickup requests and generating efficient routes to satisfy them demand a dynamic response system.

Onur Demir
Yeditepe University
E-mail: odemir@cse.yeditepe.edu.tr

Linet Özdamar
Yeditepe University
E-mail: ozdamar.linet@gmail.com

We develop a GIS based emergency response system using the hierarchical optimization guided cluster and route procedure (HOGCR) [1]. The HOGCR is adapted to accommodate special aviation constraints mentioned in [2]. We call the resulting set of algorithms "Helicopter Operations Scheduling System", (HOPS).

The primary aim of the emergency response system is to plan the routes of helicopters to satisfy delivery and pickup requests. With this goal the system collects requests with geographical references via the on-line GIS web access provided by the software ArcGIS [3], or other means of communication. These requests are converted into inputs for HOPS that generates helicopter routes to satisfy the requests. The resulting routes are illustrated on ArcGIS along with their time schedule and route/payload instruction sheets are generated for the pilots.

The post disaster response environments are somewhat chaotic, and requests change continuously over time, e.g., more injured people are brought to local shelters and new supplies are demanded. Therefore, the system has to be used in a dynamic environment where new requests are collected for a given number of periods denoted as the Request Collection Time, RCT, specified by the planner. During an RCT, the system continuously collects new requests and updates the inputs database. At each RCT, the HOPS software is re-run to obtain new routes while prioritizing the requests that are unsatisfied in the previous planning horizon (due to the limited number of available helicopters, some requests can be delayed to the next RCT).

This paper is organized as follows. Section II gives some brief information about the related work. Section III presents the work flow of the response system. In Section IV we present a test scenario, and discuss the results. Section V discusses future work and our conclusions.

2 Related Work

The use of GIS for disaster response is quite common as noted in [4], [5] and [7]. The applications of GIS cover a variety of topics including logistics, resource management, and damage estimation. As our response system, the use of GIS for evacuation planning has attracted many studies such as [7] and [8]. Usage of space technology for disaster management is studied in [9] and [10]. Another application of GIS for disaster response is using it to estimate damage as in [11], [12], [13] and [14]. Crisis management using geoinformation is another topic which is covered in [15].

3 The Response System

In this section we describe the overall structure of the proposed disaster response system. First we explain the work flow indicating the major tasks in the response system. Then we briefly describe the main components of the system including the geodatabase design, the routing algorithm and the input-output interface.

3.1 Work flow

The emergency response system aims to create distribution/pickup operations in every RCT periods and monitor the request satisfaction process dynamically. The system work flow is given below.

1. In the pre-disaster preparedness phase, hospital capacities, relief material warehouses, and helicopter landing locations (helipads in open areas where survivors can seek shelter) where delivery and pickup will take place are determined.

Population based vital materials demands are also estimated for each location. The system can be used in the preparedness phase, as a simulator for different helicopter availability and earthquake impact magnitudes. In the post-disaster response phase, hospitals and warehouses that have collapsed are deleted from the database. Furthermore, some shelters can be re-located. Estimates of relief material requests are re-evaluated.

2. In the post-disaster response phase, the dynamic collection of delivery and pickup requests gets started through on-line systems, mobile phones, etc. using every unaffected communication infrastructure. All requests are forwarded to the central server where the ArcGIS software rests. The planner merges the requests with preparedness phase demand estimations and decides on relief material quantities to be delivered to each shelter. The planner also makes sure that each shelter is protected by security forces before sending relief materials. Based on all inputs, the HOPS application generates routes for the helicopters for the current RCT. The length of each helicopter's itinerary cannot exceed RCT including load-unload durations. (For our experiments we have selected an 8-hour shift for RCT).
3. The instruction sheets are forwarded to every pilot on duty and the helicopters start executing the plan. However, if the number of helicopters is less than the number of sheets, then, shortest itineraries are delayed to the next RCT.
4. New requests and updates are continuously collected during the current RCT. The totals for every shelter are updated accordingly for the next RCT, the hospital receiving spaces are also reduced for the next RCT due to the deliveries in the current RCT. The warehouse inventories are reduced according to the relief materials that are sent out in the current RCT, and they are also augmented by new relief materials.
5. At the beginning of the next period HOPS generates new routes using the updated geodatabase, however, the itineraries that were delayed in the previous RCT are executed first. The principle of FIFO (First in First Out) applies to reduce waiting times.
6. The above procedure is repeated for as long as the relief operation continues.

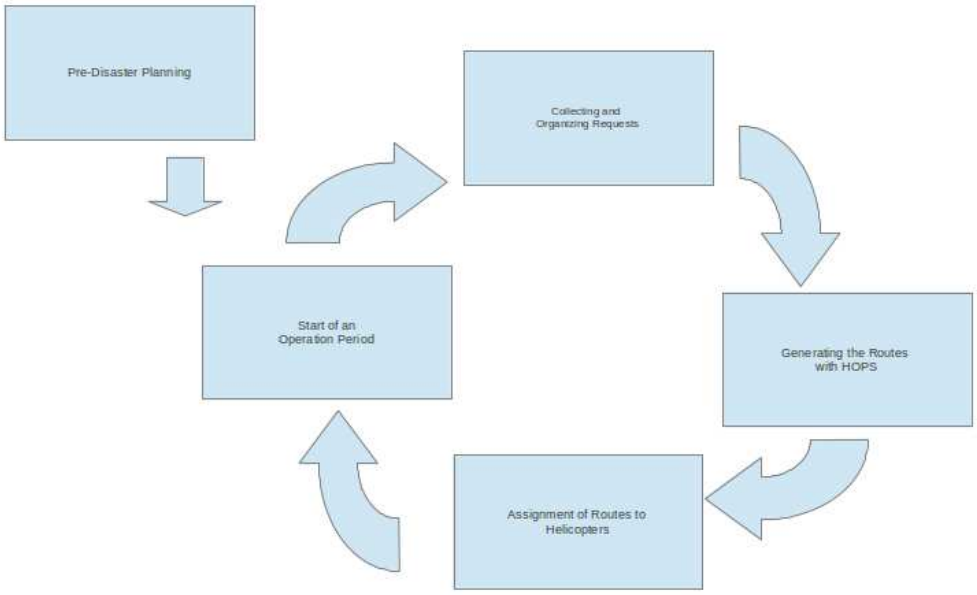


Figure 1. Work flow of the System

3.2 Components of the Response System

3.2.1 Geodatabase Design

We have used a geodatabase model to represent the requests data and their locations as well as warehouses, shelters, hospitals, etc. Geodatabase model allows the user to have relationships between the data in both logical and physical terms. (Zeiler and Zieler 2000) Each data input is identified by its location and type of the location (demand point, supply depot, hospital etc.). Each location is connected to data based on the location type. For example, shelter data contain information about total amount of demands, number of injured people, altitude and landing capacities. When required the information is pulled up from geodatabase and forwarded to the HOPS algorithm. The algorithm works externally and generates routes using exported data. In Figure 2, a sample map of Istanbul with its demand and supply points is given.

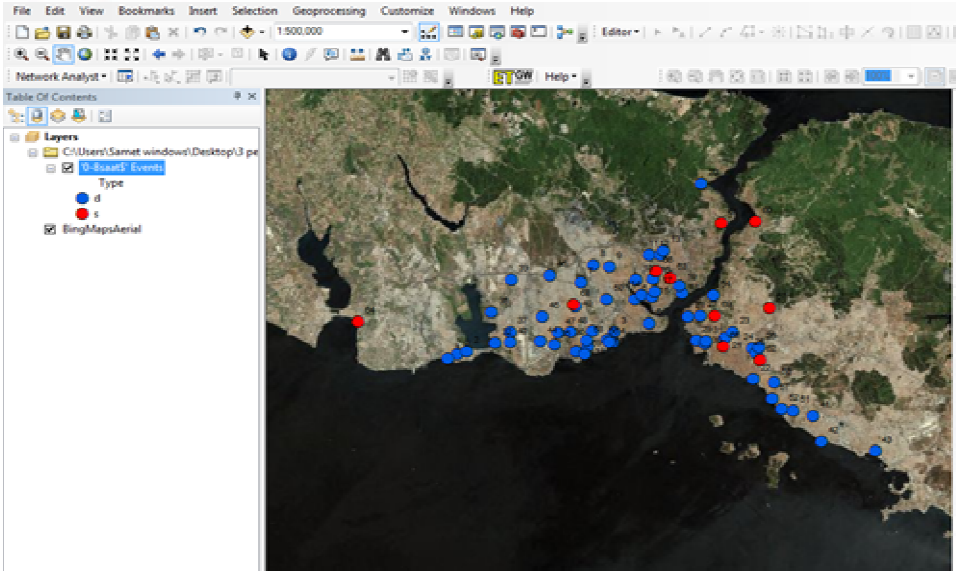


Figure 2. Administrative interface within ArcGIS (Red spots are for hospitals and blue spots are for shelter locations)

3.2.2 Administrative Interface

The response system runs on a central server. The parameters about the response system can be monitored and entered via an administrative interface on the server. The interface is used to control the parameters of HOPS algorithm. It is also used to set the parameters about helicopters, shelters, hospitals, capacities, RCT and clustering particulars.

3.2.3 HOPS Algorithm

We have programmed the HOPS algorithm in such a way that it can be configured and accessed from the administrative interface [1]. The HOPS utilizes an efficient network flow model in a hierarchical optimization guided ‘‘cluster first, route second’’ approach. We divide the demand nodes in the relief network into geographically dense clusters and then solve the top-level routing problem where the network is composed of demand cluster centers,

warehouses and hospitals. Then the flow variable values are extracted from the optimal solution of the top level problem and passed on to cluster level problems along with the updated supplier and hospital space information. Then, the routing problems of cluster networks are solved in parallel and the optimal routes obtained and merged together.The method always produces feasible solutions and it offers the flexibility of adjusting the solution quality by imposing different runtime restrictions on the optimization software package CPLEX. When it is needed, the algorithm is called and the generated routes are transferred back to the administrative interface. The routes contain stop locations and the payload of relief materials and patients in each leg of the route.

3.3 Collecting and Managing Input

We are still in the progress of providing different methods for receiving inputs. During a major disaster a damaged communication infrastructure may pose obstacles, therefore an emergency response system should not rely on a single manner of communication. We are in the process of creating an input interface for Android platforms and the web.

4. Test Scenario

The application is assessed with a scenario that takes place in post-earthquake Istanbul. An emergency relief network of 60 helipads placed in large parks and squares is considered for shelters and 10 clustered hospitals in Istanbul take place in the network. The scenario is similar to the one presented in [2], however it does not include delivery of relief materials. Rather, only the evacuation of injured people is taken into account due to the urgency of the requests. There are 20 helicopters available for the pickup operation. We simulate the operation for 24 hours that includes three 8-hour shifts during which new pickup requests arrive. The locations and the capacities of hospital clusters are predetermined. We assume that during the first, second and third 8-hour shifts, pickup requests arrive for 5925, 4489 and 11874 patients, respectively, from 60 shelters.

Initially, the total emergency patient capacity of all hospitals is 18272. After the first 8-hour shift is completed, the latter capacity is reduced to 12347, and after the second 8-hour shift, it is reduced to 7858.

	First Shift	**Second Shift**	**Third Shift**
Total Flight time including stops and re-fueling (hours)	190.11	152.13	234.71
Total number of route segments before merging	114	87	153
Average route segment length (hours)	1.67	1.75	1.53
Maximum and minimum route segment length (hours)	3.00-1.50	3.00-1.50	2.00-1.00
Total flight time after merging (hours)	190.13	152.15	234.76
Total number of merged itineraries	26	22	32
Average flight time per helicopter (hours)	7.31	6.92	7.34
Maximum-minimum flight Time (hours)	7.51-6.00	7.51-4.5	7.51-3.00
Total computation time (seconds)	129.18	40.48	128.95

Table 1. The planned itineraries

A summary of the three shift plans is presented in Table 1. In Table 1, the total flight times are displayed before and after merging route segments into itineraries of 8 hours. We merge short route segments in order to construct helicopter itineraries of a desired length. This is to reduce the number of itineraries according to the number of available helicopters. It is assumed that re-fueling can only be performed at the fuel stations next to hospitals. Merging route segments slightly increases the total flight time because of the connecting flights between

the start and end locations of two consecutive route segments. Ideally the longest itinerary should last exactly 8 hours, but this is not always the case due to the flight distances in the network topology. In our network topology, the longest itinerary turned out to be 7.5 hours. The shortest itinerary lasts 3 hours in the last shift due to lack of requests. Merging route segments naturally decreases the number of itineraries, hence, the number of helicopters needed. Yet, the number of itineraries is still larger than the available number of helicopters. This leads to the delay of some itineraries to the next shift. For example, there are 6 itineraries that could not be executed in the first shift.

The total computation time for generating plans in each shift takes less than two minutes in a 2.5 GHz Intel Quadcore Xeon server. The planned itineraries are displayed on a map in ArcGIS application in Figure 3.



Figure 3. The routes for the first 8-hour shift

5. Conclusions and Future Work

We have presented an emergency response system for post disaster delivery and pickup operations carried out by helicopters. The response system presented in the paper takes the inputs, organizes them, and generates routes for helicopters to satisfy on-going requests. Our test scenario shows that the response system can work under dynamic conditions where there are frequent information updates and the routes that are generated are quite efficient with regard to vehicle utilization and flight time. Vehicle utilization is a key component in helicopter operations due to their limited availability for civilian purposes.

Our future work is concentrated on developing other I/O applications and even faster algorithms.

Acknowledgements: We thank The Scientific and Technological Research Council of Turkey (TUBITAK) who partially supported this study with the research Grant No. 110M578.

References

1. L. Özdamar, O. Demir, A hierarchical clustering and routing procedure for large scale disaster relief logistics planning, Transportation Research Part E: Logistics and Transportation Review, Volume 48, Issue 3, May 2012, Pages 591-602, ISSN 1366-5545, 2012.
2 L. Ozdamar, Planning helicopter logistics in disaster relief , OR Spektrum, Vol. 33 (3), 655-672, 2011.

3. ArcGIS Resource Center: http://resources.arcgis.com/content/web-based- help
4. D.S.K. Thomas, S.L. Cutter, M.E. Hodgson, M. Gutekunst, and S. Jones, Use of Spatial Data and Technologies in Response to the September 11 Terrorist Attack. Boulder, CO, University of Colorado, Natural Hazards Research and Applications Information Center, Quick Response Bulletin No 153,2002.
5. B. Cahan and M. Ball. GIS at Ground Zero: Spatial technology bolsters World Trade Center response and recovery. GEOWorld 15(1): 26 –9, 2002.
6. A.J. Bruzewicz A J. Remote sensing imagery for emergency management. In Cutter S L, Richardson D B, and Wilbanks T J (eds) Geographical Dimensions of Terrorism. New York, Routledge: 87–97, 2003.
7. Y. Bo, W. Yong-gang and W. Cheng. A Multi-agent and GIS Based Simulation for Emergency Evacuation in Park and Public Square," Computational Intelligence and Security, 2009. CIS '09. International Conference on , vol.1, no., pp.211-216, 11-14 Dec. 2009.
8. B. Huang, and X. Pan, GIS coupled with traffic simulation and optimization for incident response, Computers, Environment and Urban Systems, Volume 31, Issue 2, pp.116-132, March 2007.
9. M. J. Rycroft. From environmental risks, as problems, to earth observations as solutions to those problems. Surveys in Geophysics, 21:115–125, 2000.
10. V. Jayaraman, M. G. Chandrasekhar, and U. R. Rao. Managing the natural disasters from space technology inputs. Acta Astronautica, 40(2–8):291–325,1997.
11. D.Z. Sui. Remote Sensing-Based Damage Assessment for Homeland Security , Geospatial Technologies and Homeland Security , pp.125-169, 2008.
12. S. Mansor, M. Abu Shariah, L. Billa, I. Setiawan, and F. Jabar. Spatial technology for Natural risk management. Disaster Prevention and Management, 13(5):364–373, 2004.
13. T. Ayala-Silva, Y. A. Twumasi. Hurricane Georges and vegetation change in Puerto Rico using AVHRR satellite data, International Journal of Remote Sensing, Vol. 25, Iss. 9, 2004.
14. P. Chen, S. C. Liew, and L. K. Kwoh. Tsunami damage assessment using high resolution satellite imagery: A case study of Aceh, Indonesia. In IEEE International Geoscience and Remote Sensing Symposium, July 25–29, pp. 1405–1408,2005.
15. G. Cai, A. MacEachren, R. Sharma, I. Brewer, S. Fuhrmann, and M. McNeese. Enabling geocollaborative crisis management through advanced geoinformation technologies. In dg.o2005: Proceedings of the 2005 national conference on Digital government research, pp. 227–228. Digital Government Research Center, 2005.
16. M. Zieler, M. Zeiler. Modeling our world: the ESRI guide to geodatabase design. Vol. 40. ESRI press, 2000.

MISTA 2013

Temporal Linear Relaxation in IBM ILOG CP Optimizer

Philippe Laborie · Jérôme Rogerie

Abstract IBM ILOG CP Optimizer is a constraint solver that implements a model-and-run paradigm. For scheduling problems CP Optimizer provides a relatively simple but very expressive modeling language based on the notion of interval variables. This paper presents the Temporal Linear Relaxation (TLR) used to guide the automatic search when solving scheduling problems that involve temporal and resource allocation costs. We give the rationale of the TLR, describe its integration in the automatic search of CP Optimizer and present the relaxation of most of the constraints and expressions of the model. An experimental study on a set of classical scheduling benchmarks shows that using the TLR is essential for problems with irregular temporal costs and generally helps for problems with resource allocation costs.

1 Introduction

IBM ILOG CP Optimizer is a general constraint solver that implements a model-and-run paradigm. This paper focuses on scheduling problems for which CP Optimizer provides a relatively simple but very expressive modeling language based on the notion of interval variables [12,13]. Real world scheduling applications often require optimizing irregular temporal costs (such as minimizing earliness/tardiness, minimizing or maximizing activities duration or delays between activities, *etc*) as well as non-temporal costs (resource allocation, non execution, *etc*). For these problems, even in the absence of resource constraints, providing a good time placement of activities can be a challenge. In this context, using a linear relaxation of the problem solved with Mathematical Programming techniques is a common approach. MP and CP are generally coupled in two ways. In *decomposition* approaches an MP model communicates with a distinct CP model (Sequential models, Column Generation, Logical Bender's Decomposition). Although very efficient on particular scheduling problems, these approaches

Philippe Laborie
IBM, Software Group
E-mail: laborie@fr.ibm.com

Jérôme Rogerie
IBM, Software Group
E-mail: rogerie@fr.ibm.com

tend to be difficult to generalize. In *hybrid* approaches like [3,8], the linearization is computed at each search node. For those approaches, scaling to large models may be an issue. This paper presents the approach implemented in CP Optimizer that lies somewhere in between and aims at providing both generality and scalability.

After recapping the CP Optimizer model for scheduling in section 2, we give an overview of the architecture of the automatic search for scheduling problems in section 3. The main purpose of this section is to show how the proposed Temporal Linear Relaxation (TLR) is integrated in the search. Section 4 describes the linearization of most of the scheduling concepts of the model in the TLR. The final section presents an experimental evaluation that shows the crucial positive impact of the TLR on the performance of the automatic search.

2 CP Optimizer Model for Scheduling

CP Optimizer extends classical constraint programming by introducing additional mathematical concepts (such as intervals [12], sequences or functions [13]) as new variables or expressions to capture the temporal aspects of scheduling.

An *interval variable* a is a decision variable whose domain is a subset of $\{\perp\} \cup \{[s,e) : s,e \in Z \wedge s \leq e\}$. A boolean $x(a)$ denotes the presence status of the interval variable. At a solution:

- if a is present, the value of a is $a = [s,e)$ and the presence status $x(a)$ is true. In this case, by a small abuse of notations $s(a)$, $e(a)$ and $l(a)$ respectively denote the start value s, the end value e and the length $(e-s)$ of interval variable a.
- else, a is absent, $a = \perp$ and the presence status $x(a)$ is false.

Absent interval variables have special meaning. Informally speaking, an absent interval variable is not considered by any constraint or expression on interval variables it is involved in. For example, if an absent interval variable a is used in a precedence constraint between interval variables a and b then, the constraint becomes inactive - in particular, interval variable b is not constrained by it. Each constraint and expression specifies how it handles absent interval variables.

An *optional* interval variable is an interval variable whose presence/absence status needs to be decided by the engine that is, such that singleton $\{\perp\}$ is strictly included in the domain. The use of optional interval variable is quite versatile. A typical use is for expressing an optional activity in a schedule. Another common use is for modeling a task requiring one machine among an alternative machine set. For each machine candidate an optional interval variable models the possible execution of the task on the machine. Then the solver will decide which one among those optional interval variables is present (that is, which machine is selected), the others being absent. By nature interval variables are optional and they can be made non-optional by adding constraints which remove $\perp$ from the domain.

The following constraints on interval variables are introduced to model the basic structure of scheduling problems. Let a, a_i and b denote interval variables and z an integer variable:

- A *presence constraint* presenceOf(a) states that interval a is present, that is $a \neq \perp$. This constraint can be composed, for instance presenceOf(a) $\Rightarrow$ presenceOf(b) means that the presence of a implies the presence of b.

- A *precedence constraint* (*e.g.* endBeforeStart(a, b, z)) specifies a precedence between interval end-points with an integer or variable minimal distance z provided both intervals a and b are present.
- A *span constraint* span $(a, \{b_1, ..., b_n\})$ states that if a is present, it starts together with the first present interval in $\{b_1, ..., b_n\}$ and ends together with the last one. Interval a is absent if and only if all the b_i are absent.
- An *alternative constraint* alternative $(a, \{b_1, ..., b_n\})$ models an exclusive alternative between $\{b_1, ..., b_n\}$: if interval a is present then exactly one of intervals $\{b_1, ..., b_n\}$ is present and a starts and ends together with this chosen one. Interval a is absent if and only if all the b_i are absent.

These constraints make it easy to capture the hierarchical, logical and temporal structure of scheduling problems (the breakdown structure of a project, optional activities, alternative modes/recipes/processes, etc.) in a well-defined CP paradigm [12].

CP Optimizer also provides numerical expressions related with the integer bounds (start, end, length) of an interval variable. These expressions are shown on Table 1. The length of an interval variable is the distance between its start and end points. In these expressions, F is a piecewise linear function. For instance $endEval(a, F, V)$ could be used to model an earliness/tardiness cost function F for an optional activity modeled by an interval variable a evaluated on its end time while value V represents a non-execution cost for the activity. A wide range of arithmetic expressions and constraints are available in CP Optimizer to combine these numerical expressions.

startOf(a, V)	$s(a)$ if $x(a)$ is true V otherwise
endOf(a, V)	$e(a)$ if $x(a)$ is true V otherwise
lengthOf(a, V)	$l(a)$ if $x(a)$ is true V otherwise
startEval(a, F, V)	$F(s(a))$ if $x(a)$ is true V otherwise
endEval(a, F, V)	$F(e(a))$ if $x(a)$ is true V otherwise
lengthEval(a, F, V)	$F(l(a))$ if $x(a)$ is true V otherwise

Table 1: Expression semantics

For modeling resources, and more generally any aspect of the problem involving a timeline, CP Optimizer offers concepts such as *cumul functions*, *state functions* and *interval sequences* [13].

We illustrate how timelines are relaxed in the TLR in section 4 for the particular case of *cumul functions*. A cumul function is an expression whose value in a solution is an integer stepwise function. A cumul function is defined as the sum of individual contributions of interval variables. For instance, an interval variable a_i could contribute to a cumul function $f = \sum_i pulse(a_i, h_i)$ with a pulse function $pulse(a_i, h_i)$. When interval variable a_i is present, a pulse function $pulse(a_i, h_i)$ is a step function equal to value h_i between the start and end of interval a_i and equal to value 0 outside the interval. Otherwise, if the interval variable is absent, it is the 0 function. Constraints can be posted on cumul functions to limit their allowed minimal or maximal value

over the complete horizon ($f \leq K$) or over some constant or variable intervals. Cumul functions are typically used to model cumulative resources.

The combination of constraints on the above timeline expressions together with structural constraints on interval variables like *span* and *alternative* facilitate the modeling of complex resources.

3 Integration of TLR in CP Optimizer Automatic Search.

Large Neighborhood Search (LNS) [20] is a component of CP Optimizer automatic search for scheduling. It consists of a process of continual relaxation and re-optimization: a first solution is computed and iteratively improved. Each iteration consists of a relaxation step followed by a re-optimization of the relaxed solution. This process continues until some condition is satisfied, typically, when the solution can be proved to be optimal or when a time limit is reached. In CP Optimizer this approach is robustified by using portfolios of large neighborhoods and completion strategies in combination with Machine Learning techniques to converge on the most efficient neighborhoods and completion strategies for the problem being solved.

In this context, the large neighborhoods are all based on the initial generation of a Partial Order Schedule (POS) [17] constructed from a completely instantiated solution where interval variables have fixed start and end values. A POS is a directed graph G($\mathcal{A}$, $\mathcal{E}$) where the edges in $\mathcal{E}$ are precedence constraints between interval variables with the property that any solution to the graph is also a resource-feasible solution that is, it satisfies the constraints posted on *interval sequence variables*, *cumul expressions* and *state function variables*. Algorithms for transforming a fully instantiated solution into a POS P are described in [10,12]. Large neighborhoods in the portfolio differ in the way they select a subset of interval variables to be relaxed (fragment). The relaxed POS P' is obtained by removing from P all the edges involving at least one selected interval variable and adding new edges to repair broken paths. The relaxed POS P' is then used to enforce precedence constraints between interval variables before applying a completion strategy.

The completion strategies partially explore a search tree with a limited number of backtracks. At the root node of an LNS iteration, in case of irregular temporal costs, an important issue is finding a good time placement for the interval variables in the LNS fragment considering the frozen part of the POS. In this context some of the strategies solve the TLR described in this paper. The optimal solution of this TLR gives indicative start and end values for each interval variable. The search is a generalization of the *SetTimes* strategy recapped in [10]. It considers interval variables by increasing indicative end point values and tries in the left branch to set them as close as possible to their indicative values. When a failure occurs in the left branch, the interval variable is marked *unselectable* and will remain so until constraint propagation removes the value tried in the left branch from the domain of the variable. When the objective is regular, this strategy boils down to *SetTimes*.

The objective function is automatically analyzed before solving the problem and the completion strategy portfolio is configured as follow:

- for objective functions involving some irregular temporal terms (like earliness cost or interval length maximization), only completion strategies using the TLR are used,

– for objective functions involving terms related with presence of interval variables (like resource allocation costs) but only regular temporal objective, both strategies using the TLR and the *SetTimes* are used, in this case the learning techniques will favor the most efficient strategy on the instance being solved,
– for other objective functions (no presence-related terms and no irregular temporal terms), only strategies based on *SetTimes* are used.

Note that the TLR is also used at the root node of the global search tree to compute a lower bound on the cost.

4 Temporal Linear Relaxation Formulations

The TLR is a continuous Linear Program that represents the relaxation of the current problem solved by the engine at the root node of an LNS iteration. In CP Optimizer, this TLR is solved using CPLEX. This section describes the relaxation of most of the elements of the model.

4.1 Interval Variables and Logical Constraints

An interval variable a_i is represented by four numerical variables in the TLR: three temporal variables s_i, e_i and l_i for the start point, end point and length together with one continuous $[0, 1]$ variable x_i for the presence status. The range of temporal variables s_i, e_i and l_i is the domain of the interval start, end and length in the CP engine. The constraint on the length of interval a_i can be seen as a special cases of precedence constraint with delay between the start and the end of the interval, so it will be described in section 4.2.

In the CP engine, binary logical constraints like $\text{presenceOf}(a_i) \Rightarrow \text{presenceOf}(a_j)$ are aggregated into an implication network that is updated from the model statements and from engine deductions [12]. The TLR formulation of these constraints consists of binary linear inequalities like $x_i \leq x_j$.

4.2 Precedence Constraints

Precedence constraints are constraints like $\text{endBeforeStart}(a_i, a_j, z_{ij})$ where z_{ij} is a delay. Delay z_{ij} can be constant or variable. These constraints and the constraints on interval lengths are aggregated by CP Optimizer in a temporal constraint network [12]. A precedence relation is a constraint of the form $(x_i \wedge x_j) \Rightarrow (y_i + z_{i,j} \leq y_j)$ where x_i and x_j are the presence status of intervals a_i and a_j and y_i, y_j the interval endpoints related with the precedence constraint (s_i or e_i, s_j or e_j). Let $M_i^{i,j}$ and $M_j^{i,j}$ denote large enough coefficients, the LP formulation reads:

$$y_i + z_{i,j} \leq y_j + M_i^{i,j}(1 - x_i) + M_j^{i,j}(1 - x_j)$$

Note that in the presence of logical relations between intervals, this formulation can be simplified. For instance if $x_i \Rightarrow \neg x_j$, the precedence constraint does not need to be linearized at all because it is inactive. If the presence status of interval variables is equivalent, that is $x_i \Leftrightarrow x_j$, a single big-M coefficient can be used. This is in particular

the case for precedence constraints representing the length of interval variables as, here, y_i and y_j are endpoints of the same variable.

Let $[\underline{Y_i}, \overline{Y_i}]$ denote the domain of y_i. In order to get a tighter linear relaxation, the values of big-M coefficients are minimized by selecting a specific absence value $Y_i^0 \in [\underline{Y_i}, \overline{Y_i}]$ when the interval is absent ($x_i = 0$). The following constraint is added to fix y_i to value Y_i^0 when $x_i = 0$:

$$Y_i^0 + (\underline{Y_i} - Y_i^0)x_i \leq y_i \leq Y_i^0 + (\overline{Y_i} - Y_i^0)x_i$$

Under these conditions, if $[\underline{Z_{i,j}}, \overline{Z_{i,j}}]$ denotes the domain of $z_{i,j}$ for the general case we can select:

$$M_i^{i,j} = \max\left(0, Y_i^0 + \overline{Z_{i,j}} - \underline{Y_j}\right), M_j^{i,j} = \max\left(0, \overline{Y_i} + \overline{Z_{i,j}} - Y_j^0\right)$$

Absence values Y_i^0 are selected so as to minimize the big-M coefficients resulting in a tighter relaxation. For instance if $x_i = x_j$, $Z_{i,j}$ constant, and $Y_i^0 + Z_{i,j} \leq Y_j^0$, the big-M value is zero.

4.3 Alternative Constraints

An alternative constraint $\mathit{alternative}\,(a, \{b_1, ..., b_n\})$ is both an exclusive disjunction of the presence status of the b_i's and start and end points synchronization between a and the b_i's (precedence constraints startAtStart(a, b_i), endAtEnd(a, b_i)). The precedences are added to the temporal network. Beside the relaxation of those precedences (see §4.2), the constraint $x(a) = \sum_{i \in \{1..n\}} x(b_i)$ is added to TLR.

4.4 Span Constraints

A span constraint $\mathit{span}\,(a, \{b_1, ..., b_n\})$ is a conjunction of:

- a set of precedences b_i starts after and ends before a that are linearized as described in §4.2.
- a set of implications $x(b_i) \Rightarrow x(a)$ that are added in the TLR ($x(b_i) \leq x(a)$)
- conditions $s(a) \geq \min_{i \in \{1..n\}} s(b_i)$ and $e(a) \leq \max_{i \in \{1..n\}} e(b_i)$ are convexified in the TLR.

4.5 Piecewise Linear Expressions on Interval Endpoints

CP Optimizer provides piecewise linear expressions evaluated on interval endpoints and lengths, for instance endEval(a, F, V^0) where F is a piecewise linear function (see Table 1). This expressions evaluates to $F(\text{endOf}(a))$ when interval variable a is present and to V^0 when the interval is absent. In the TLR, this expression is convexified over the CP domain of the interval endpoint $y = e(a)$ as illustrated on Fig. 1.

When the interval variable is optional, if v_p denotes the convexified domain in the TLR, x and y respectively the presence and temporal variables of the interval of absence value of Y^0, the expression is linearized as:

$$v_p + (1 - x)(V^0 - F(Y^0))$$

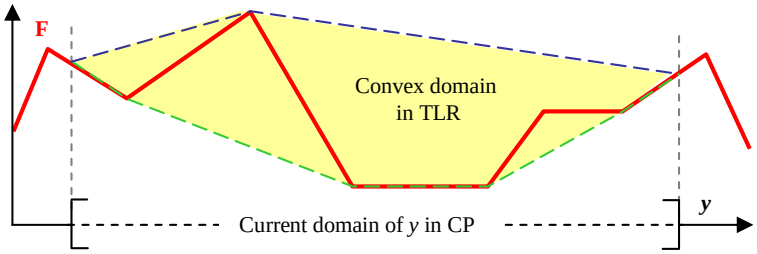


Fig. 1: Temporal relaxation of piecewise-linear expressions

More generally, all the expressions over interval variables in CP Optimizer involve some absence value V^0 that represents the value of the expression when some interval is absent [12]. Unary expressions on the ranges of an interval variable follow the same scheme as piecewise linear expressions: for example, the expression $\text{startOf}(a, V^o)$ is relaxed by $s(a) + (1 - x(a))(V^0 - S^0)$ where S^0 is the absence value of the start of a (see §4.2).

4.6 Arithmetical Expressions

CP Optimizer provides a wide range of arithmetical non-linear expressions such as $\min_i x_i$, $\max_i x_i$, xy, $1/x$, $|x|$, x^k, e^x, $\log(x)$, ...

The TLR implements some classical linear relaxations of those expressions. For instance:

- Univariate expressions are convexified in a similar way as piecewise linear expressions. For instance expressions x^{2k+1} are convexified using envelopes studied in [14],
- Bilinear terms xy assuming $0 \leq x$ and $0 \leq y$ are relaxed as $\underline{X}y \leq xy \leq \overline{X}y$ and $x\underline{Y} \leq xy \leq x\overline{Y}$

4.7 Overlap Length expressions

An *overlap length* expression $\text{overlapLength}(a_1, a_2, V^0)$ measures the length of the intersection of two intervals a_1 and a_2 as illustrated on Fig. 2 and returns value V^0 if some of the interval variables a_1 or a_2 are absent.

The overlap length is the minimum length of the four possible candidate intervals between an end point and a start point of intervals $\{a_1, a_2\}$:

$$\begin{array}{ll} \text{Candidate terms:} & \forall i, j \in \{1, 2\},\ c_{ij} = \max(0, e_j - s_i) \\ \text{Present intervals formula:} & \min(c_{11}, c_{22}, c_{12}, c_{21}) \end{array}$$

Overlap length expressions are a useful complement of precedence constraints. They can be used for measuring the partial or total temporal disjunction between two tasks or for computing the total energy required by a set of activities a_i requiring q_i units of resource over some time window b as $\sum_i q_i \, \text{overlapLength}(a_i, b)$.

The relaxation of the overlap length expression will allow to present a general scheme for the formulation in the TLR of any binary expression and constraint on

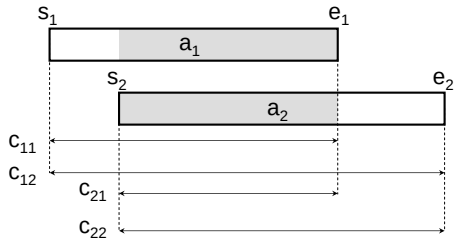


Fig. 2: Overlap length expression

interval variables. It will also illustrate how to exploit the data structure of propagation algorithms to calculate a tight convexification of the present intervals formulation.

4.7.1 Handling Presence Statuses in Binary Expressions

This section shows how similar ideas as the ones used to linearize unary expressions can be applied to binary expressions like *overlap length*.

Let's consider an expression overlapLength(a_1, a_2, V^0). The [0,1] variables in the TLR that represents the presence status of intervals a_1 and a_2 are denoted x_1, x_2 while ov_p denotes the expression calculated from interval endpoints and lengths (s_1, e_1, l_1, s_2, e_2, l_2) assuming both intervals are present. $\underline{OV}$, $\overline{OV}$ are the lower and upper bound of ov_p as computed using the current bounds in the propagation engine. OV^0 is the value of the expression as computed from the absence value of the intervals endpoints in the TLR (see §4.2). With $x_{12} = x_1 x_2$, the expression ov to relax is:

$$ov = x_{12}\, ov_p + (1 - x_{12})\, V^0$$

The formulation of these constraint in the TLR consist in relaxing $x_{12}\, ov_p$. There are two cases depending on the relation between presence statuses:

– In the general case, with x_{12} a Boolean variable and ov_p being positive:

$$\begin{aligned} \underline{OV}\, x_{12} + (1 - x_{12})\, V^0 \leq ov \leq ov_p + (1 - x_{12})\, V^0 \\ \leq \overline{OV}\, x_{12} + (1 - x_{12})\, V^0 \end{aligned} \tag{1}$$

– When both interval variables a_1 and a_2 are absent, variable ov_p will take the absence value OV^0. This can be exploited to tighten the linear relaxation depending on the logical relations in the logical network. In such cases, an equation similar with the unary expression case is obtained.

$$ov = ov_p + (1 - x_1)(V^0 - OV^0) \tag{2}$$

There are four possible formulations for the expression in the TLR.

Logical relation	Presence variable	TLR formulation
if $x_1 \Rightarrow \neg x_2$	$x_{12} = 0$	$ov = V^0$
if $x_1 \Leftrightarrow x_2$	$x_{12} = x_1$	(1)
if $x_1 \Rightarrow x_2$	$x_{12} = x_1$	(2)
otherwise	$x_{12} \leq x_1,\ x_{12} \leq x_2$ $x_1 + x_2 \leq x_{12} + 1$	(2)

The formulation (1) is also used when the involved ranges of one of the interval are fixed. The case where both intervals are present ($x_{12} = 1$) is also covered by formulation (1).

This formulation is a general scheme that can be applied to any binary expression or constraint. We will now see how the present interval formula ov_p is linearized in the case of the *overlap length* expression.

4.7.2 Convexification of Present Interval Formula ov_p

We have seen that ov_p can be formulated as the minimal value of four candidate terms $ov_p = \min(c_{11}, c_{22}, c_{12}, c_{21})$. Terms c_{ii} are the length of interval a_i, they already have a corresponding term l_i in the TLR. Terms c_{12} and c_{21} measure how much a_1 precedes or follows a_2. Their linearization in the TLR is the one of the $\max(0, e_j - s_i)$ expression.

Upper bound formulation of ov_p. The upper bound of ov_p is formulated by stating constraints $ov_p \leq c_{ij}$ for all $i, j \in \{1, 2\}$. Good practices in propagation consist in memorizing and incrementally maintaining supports for bounds on range involved in a constraint or an expression. By support, we mean a piece of information that can be updated in reasonable time and that avoids further calculations. For the upper bound of the overlap expression, with $\mathcal{C} = \{c_{ij}, i, j \in \{1, 2\}\}$, the support $\mathcal{S}$ is a minimal set of candidates required to calculate the upper bound of ov_p. Initially, we set $\mathcal{S} = \mathcal{C}$. If $\underline{C}$ and $\overline{C}$ respectively denote the lower and upper bounds on a candidate c:

$$\forall c \in \mathcal{S},\ \left\{\exists\ c' \in \mathcal{S} \setminus \{c\} \mid \overline{C}' \leq \underline{C}\right\} \Rightarrow c \notin \mathcal{S}$$

Maintaining $\mathcal{S}$ is of interest if the constrained structures involving the variables allow to reduce its size. Here are some typical usages of $\mathcal{S}$ in the propagation engine and in the TLR:

- $\mathcal{S}$ *is empty*: the propagation engine has found an inconsistency.
- $\mathcal{S}$ *is a singleton* $\{c\}$: the propagation engine filtering rule is $\overline{OV} = \overline{C}$ and the constraint $c = ov_p$ is added to the TLR.

Convexification of the lower bound of ov_p. By considering a minimal set $\mathcal{S}$, the propagation may achieve stronger domain reductions on a range of the intervals. Eventually, $\mathcal{S}$ allows to improve the convexification of ov_r. Fig. 3 illustrates the idea of this convexification. We suppose $|\mathcal{S}| > 1$. Be c and c' the two candidates in $\mathcal{S}$ such that $\underline{C}$ is the minimal value of the overlap length ($\underline{OV} = \underline{C}$) and $\underline{C}'$ the next minimal value of the overlap length.

Suppose we have $\underline{C} < \underline{C}'$. Necessarily, $\underline{C}' < \overline{OV} \leq \overline{C}$: otherwise c would be the only candidate in $\mathcal{S}$. Consequently, an upper bound inequality can be stated as:

$$(\overline{C} - \underline{OV})ov_p \geq (\underline{OV}' - \underline{OV})c + \underline{OV}(\overline{C} - \underline{OV}')$$

Removing candidates of same or near lower bound strengthen the relaxation. Next paragraph show how precedences may remove candidates.

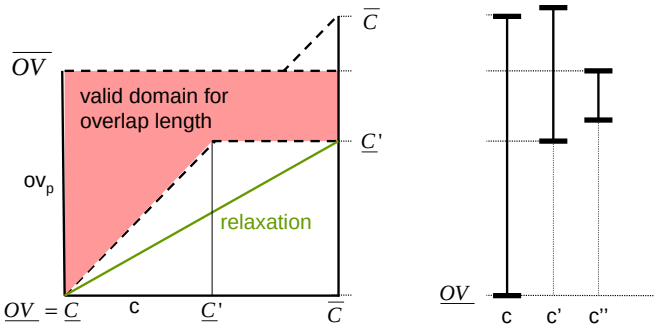


Fig. 3: Overlap length relaxation of lower bound

Candidate elimination . The propagation engine uses two kinds of argument to compute and incrementally maintain a set $\mathcal{S}$.

– *Propagation of the constraint network on the expression*: when a new upper bound ub of the overlap length is propagated, the following rule applies:

$$\forall c \in \mathcal{S},\ \left(\exists c' \in \mathcal{S} \setminus \{c\}, \min(\overline{C}', ub)) \leq \underline{C}\right) \Rightarrow c \notin \mathcal{S}$$

– *Precedence network*: if, from the precedence network or the actual bound of the range of intervals, $e_1 \leq e_2$ then $c_{11} = e_1 - s_1 \leq e_2 - s_1 \leq c_{21}$. That is c_{21} can eventually be removed from $\mathcal{S}$. The elimination rules is:

$$\forall i,j \in \{1,2\} \begin{cases} s_i \leq s_{j\neq i} \Rightarrow \begin{cases} c_{1j} \in \mathcal{S} \Rightarrow c_{1i} \notin \mathcal{S} \\ c_{2j} \in \mathcal{S} \Rightarrow c_{2i} \notin \mathcal{S} \end{cases} \\ e_i \leq e_{j\neq i} \Rightarrow \begin{cases} c_{i1} \in \mathcal{S} \Rightarrow c_{j1} \notin \mathcal{S} \\ c_{i2} \in \mathcal{S} \Rightarrow c_{j2} \notin \mathcal{S} \end{cases} \end{cases}$$

In CP Optimizer, the TLR is computed after restoration of the relaxed POS. The relaxed POS contains many precedence relations that strongly benefits the candidate elimination.

4.8 Timelines and their Constraints

Resources and more generally in CP Optimizer, timelines and their related constraints, are often discrete, sometimes disjunctive and generally non-linear concepts. That is one of the reasons why scheduling problems are often difficult to solve with MP techniques alone.

By construction of the POS in the LNS, timelines values are represented as a set of precedence constraints. Those precedence constraints give flexibility for re-inserting the interval variables of the relaxed LNS fragment. They also fit well in the TLR because they are linear constraints (see §4.2). In this context, the TLR solved at the root node of an LNS iteration will provide guidelines for re-positioning the relaxed interval variables in the frozen part of the temporal network.

A very tight relaxation of the timelines constraints is not crucial here because most of those constraints are already linearized as precedence constraints in the POS. This

being said, the TLR includes some global linear constraints related with timelines. For instance for a maximum value of a cumul function like: $f = \sum_i pulse(a_i, h_i) \leq K$, if T_{min} and T_{max} respectively denote the minimal start value and maximal end value of all possibly present intervals a_i then the following constraint must hold:

$$\sum_i l_i \, h_i \leq K(T_{max} - T_{min})$$

This type of energetic constraints are added in the TLR for each cumul function.

5 Experimental Evaluation

We studied the impact of the TLR on 15 scheduling benchmarks involving irregular temporal cost functions (T) and/or interval presence related costs (P):

1. Oversubscribed satellite communication scheduling problems [11] (P)
2. Job-shop scheduling problems with earliness/tardiness costs [1] (T)
3. Audit scheduling problems [6] (P)
4. Multi-machine assignment scheduling problem [5] (P)
5. Single machine instances of the Masc library [16] (T)
6. Personal tasks scheduling [19] (T&P)
7. RCPSP with discounted cash flows [21] (T)
8. Single machine scheduling problems with earliness/tardiness costs [7] (T)
9. Flow-shop scheduling problems with earliness/tardiness costs [15] (T)
10. A non-public scheduling problem involving earliness/tardiness costs (T)
11. Aircraft landings scheduling [2] (T)
12. Common due-date scheduling problems [4] (T)
13. Dynamic resource feasibility problems [9] (T)
14. Maximal quality RCPSP [18] (T)
15. RCPSP with earliness/tardiness costs [22] (T)

Experiments were performed with CP Optimizer V12.5. For each benchmark we estimate the speed-up factor of TLR as the ratio of the time it takes to find a solution of similar quality for a version of CP Optimizer not using the TLR and for the default version of CP Optimizer (using the TLR). Note that in CP Optimizer, the TLR can be switched off by setting parameter `TemporalRelaxation` to value `Off`. The results are summarized on Fig. 4. On the figure, the speed-up factor is capped to 100. On benchmarks with speed-up greater than 100, we noticed that even the initial solution using the TLR could never be reached within the time-limit when not using the TLR.

Three categories of benchmarks are considered: problems with irregular temporal costs, problems with presence-related cost and problems with both types of costs.

The experimental evaluation clearly shows that the TLR is essential for problems with irregular temporal costs and generally helps for problem with presence related costs.

6 Conclusion

This paper presents the temporal linear relaxation of the CP Optimizer model and its integration in the automatic search to control the completion strategies. We think

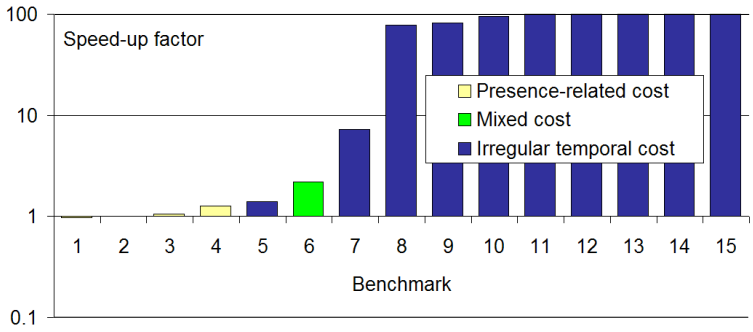


Fig. 4: Speed-up factor when using TLR

that this loose coupling between a relaxation and the CP engine at the root node of each LNS iteration is an efficient and robust method if the relaxation can exploit the dominant structures of the problem to provide well informed guidelines for the completion strategy. In the context of CP Optimizer where only a part of the POS is unfrozen, the TLR tends to be very informative as most of the resource constraints are still captured by frozen precedence arcs of the POS. For instance in the extreme case of an empty fragment where no interval variables are relaxed in the POS, the TLR boils down to the computation of the optimal placement of interval variables given the temporal network of the POS.

Future work will concern both the control and the diversification of the temporal relaxation by:

- Controlling the size and the content of the TLR depending on the characteristics of the problem and the current impact on the search,
- Exploring tighter relaxations in the TLR, for instance by solving a MILP enforcing the boolean presence status of interval variables,
- Considering concurrent relaxations that exploit other dominant structures of the problem like capacity planning or sequencing sub-problems.

References

1. P. Baptiste, M. Flamini, and F. Sourd. Lagrangian bounds for just-in-time job-shop scheduling. *Computers and Operations Research*, 35:906–915, 2008.
2. J.E. Beasley, M. Krishnamoorthy, Y.M. Sharaiha, and D. Abramson. Scheduling aircraft landings - the static case. *Transportation Science*, 34:180–197, 2000.
3. C. Beck and P. Refalo. A hybrid approach to scheduling with earliness and tardiness costs. In *Proc. 3th International CP-AI-OR Conference (CP-AI-OR 2001)*, 2001.
4. D. Biskup and M. Feldmann. Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers and Operation Research*, 28(8):787–801, 2001.
5. A. Bockmayr and N. Pisaruk. Detecting infeasibility and generating cuts for mixed integer programming using constraint programming. *Computers and Operations Research*, 33:2777–2786, 2006.
6. P. Brucker and D. Schumacher. A new tabu search procedure for an audit-scheduling problem. *Journal of Scheduling*, 2:157–173, 1999.
7. K. Bulbul, P. Kaminsky, and C. Yano. Preemption in single machine earliness/tardiness scheduling. *Journal of Scheduling*, 10:271–292, 2007.

8. E. Danna and L. Perron. Structured vs. unstructured large neighborhood search: A case study on job-shop scheduling problems with earliness and tardiness costs. In *Proc. 9th International CP Conference (CP 2003)*, pages 817–821, 2003.
9. H. el Sakkout, T. Richards, and M. Wallace. Minimal perturbation in dynamic scheduling. In *Proc. ECAI-98*, 1998.
10. D. Godard, P. Laborie, and W. Nuijten. Randomized Large Neighborhood Search for Cumulative Scheduling. In *Proc. ICAPS-05*, pages 81–89, 2005.
11. L. A. Kramer, L. V. Barbulescu, and S. F. Smith. Understanding Performance Tradeoffs in Algorithms for Solving Oversubscribed Scheduling. In *Proc. 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1019–1024, 2007.
12. P. Laborie and J. Rogerie. Reasoning with conditional time-intervals. In *Proc. 21th International FLAIRS Conference (FLAIRS 2008)*, 2008.
13. P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. Reasoning with Conditional Time-intervals, Part II: an Algebraical Model for Resources. In *Proc. 22th International FLAIRS Conference (FLAIRS 2009)*, 2009.
14. L. Liberti and C. C. Pantelides. Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25:157–168, 2003.
15. T. Morton and D. Pentico. *Heuristic Scheduling Systems*. Wiley, 1993.
16. W. Nuijten, T. Bousonville, F. Focacci, D. Godard, and C. Le Pape. Towards an industrial manufacturing scheduling problem and test bed. In *Proc. 9th Int. Conf. on Project Management and Scheduling*, 2004.
17. N. Policella, A. Cesta, A. Oddi, and S.F. Smith. Generating robust schedules through temporal flexibility. In *Proceedings ICAPS-04*, Whistler, Canada, June 2004.
18. N. Policella, X. Wang, S.F. Smith, and A. Oddi. Exploiting temporal flexibility to obtain high quality schedules. In *Proc. AAAI-2005*, 2005.
19. I. Refanidis. Managing personal tasks with time constraints and preferences. In *Proc. 17th International Conference on Automated Planning and Scheduling Systems (ICAPS-07)*, pages 272–279, 2007.
20. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proc. CP-98*, pages 417–431, 1998.
21. M. Vanhoucke. A scatter search heuristic for maximising the net present value of a resource-constrained project with fixed activity cash flows. *International Journal of Production Research*, 48:1983 – 2001, 2010.
22. M. Vanhoucke, E. Demeulemeester, and W. Herroelen. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research*, 102(1-4):179–196, 2001.

MISTA 2013

DAG Scheduling using Integer Programming in heterogeneous parallel execution environments

Christos Valouxis • Christos Gogos • Panayiotis Alefragis • George Goulas • Nikolaos Voros • Efthymios Housos

Abstract A computer program can be represented by a Directed Acyclic Graph (DAG) in order to capture the dependencies between the individual tasks that should be executed each time the program runs. This paper proposes a mathematical model of Integer Programming that can be applied in order to schedule the tasks in the presence of multiple processors serving as the execution environment. The target is to minimize the overall execution time of the DAG known as schedule length or makespan. An approach called MATH using the full model is applied to small sized problems and then a more elaborate approach called MATHL is presented where the DAG is partitioned to levels. Levels are formed according to the hops needed for each node to be reached starting from the source node. Hence sub-problems have manageable size and can be solved in a timely manner. Consecutive optimal solutions for each level result in a high quality schedule for the overall problem even for cases consisting of several hundreds of nodes. Results show that this method constantly gives very good results and it is compared favorably with other approaches to the problem that can be found in the bibliography.

Keywords process scheduling; static scheduling; directed acyclic graphs; integer programming; heterogeneous environment; heuristics

1 Introduction

Christos Valouxis
Technological Educational Institute of Mesolonghi, Greece
E-mail: cvalouxis@gmail.com

Christos Gogos
Technological Educational Institute of Epirus, Greece
E-mail: chgogos@gmail.com

Panayiotis Alefragis
Technological Educational Institute of Mesolonghi, Greece
E-mail: alefrag@teimes.gr

George Goulas
Technological Educational Institute of Mesolonghi, Greece
E-mail: ggoulas@teimes.gr

Nikolaos Voros
Technological Educational Institute of Mesolonghi, Greece
E-mail: voros@teimes.gr

Ethymios Housos
University of Patras, Greece
E-mail: housos@ece.upatras.gr

A computer application might speed its execution on a multiple processor system over a single processor one. This premise seems to motivate researchers on their quest to find automatic ways of transforming applications originally coded for a single processor to equivalent ones that can be executed on parallel environments. An application consists of subtasks and distribution of them to execute on different processors should adhere to certain control and data dependencies imposed by the structure of the application. Scheduling an application consists of assigning subtasks to processors and defining the order of execution among them. Efficient scheduling has tremendous impact on execution speed.
Applications are commonly represented as Directed Acyclic Graphs (DAGs) so as to capture dependencies and communications among tasks. In a DAG each node has a value and represents a task and its computation cost, while directed arcs between nodes represent the communication cost from the source task to the target task. Typically, an application undergoes decomposition and dependence analysis in order to be converted to a DAG.
The general form of scheduling a task graph is a documented NP-hard problem [1]. Therefore, optimal solutions for complex graphs are not expected to be found in polynomial bound time, unless P = NP. In practice, several approximation techniques are used aiming to produce near optimal solutions within acceptable time limits.
The contribution of this paper is the proposal of a mathematical model for the problem of scheduling a DAG in a heterogeneous parallel execution environment. Experiments demonstrate that very good results can be obtained in reasonable time for graphs with hundreds of tasks. Comparisons with state of the art techniques further exhibit the strength of the approach.
The rest of the paper is structured as follows: Section 2 introduces the problem alongside with major approaches found in the literature. Section 3 presents the proposed mathematical model and Section 4 describes the decomposition of the problem to subproblems of manageable size. In Section 5 experiments are presented on randomly generated and non-randomly generated task graphs demonstrating benefits of the approach. Finally, Section 6 presents conclusions and future work.

2 Problem definition and related work

The motivation behind this work is to achieve shortest possible execution times for computer applications in the presence of a multi processor environment. The target system is considered to consist of a set of fully connected heterogeneous processors together with a dedicated communication subsystem. It is assumed to be contention free, capable of unlimited concurrent communications without the involvement of the processors. Therefore, computations can overlap with communications freely. In most of the pragmatic cases, parallel systems are not contention free nor are the processors uninvolved to the communications. Nevertheless, the above assumptions are common in studying task scheduling approaches ahead of applying them to more accurate models of parallel execution environments.
The computer application is assumed to be in the form of a node and edge weighted DAG with each node representing a task and each directed edge representing communication needed between tasks. Each task can start execution if and only if it receives data from all its predecessors. The computation cost of each task in each processor, since the target system is heterogeneous, is given alongside with the data transfer cost between tasks that occur only when a pair of connected tasks are scheduled to execute on different processors. Otherwise, since data are communicated over the same processor the delay is considered to be negligible. Task execution once started is assumed to finish in nonpreemptive manner. The aim is to minimize the total execution length of the application while respecting precedence constraints and resource availabilities. It is presumed that the execution time of each task and the amount of data communicated between tasks are known in advance and are static.
Research on task scheduling for homogeneous systems preceded investigation of the same problem for heterogeneous systems. Several scientific articles can be consulted for both problem specifications [2] alongside with review papers [3], [4] and books [5]. The major

categories of task scheduling techniques on heterogeneous systems are list scheduling, clustering, node duplication algorithms and stochastic search algorithms. List scheduling is the most frequently cited technique for task scheduling. Nodes are ordered according to a priority scheme, obeying precedence constraints and then each node is scheduled to a processor chosen among available processors. Some list scheduling heuristics are Earliest Time First (ETF) [6], Critical Path on a Processor (CPOP) and Heterogeneous Earliest Finish Time (HEFT) [7]. In particular, HEFT is considered to be one of the most accepted heuristics for task scheduling since it is very fast and produces good quality schedules most of the times. HEFT assigns a ranking value to each node ($rank_u$) by traversing upwards the DAG. The value of $rank_u$ for each node n_i is defined as follows:

$$rank_u(n_i) = \overline{w_i} + max_{n_j \in succ(n_i)} \left(\overline{c_{ij}} + rank_u(t_j) \right), \quad (1)$$

where $\overline{w_i}$ is the average execution time of node n_i over all processors and $\overline{c_{ij}}$ is the average communication cost between nodes n_i and n_j. Nodes are prioritized based on $rank_u$ values giving higher priority to larger values. Then, each node is scheduled to the processor that finishes execution of it earlier than the others. HEFT popularity leaded to variations of the basic algorithm like HEFT using Look Ahead (HEFTLA) [8].

The second major task scheduling category of techniques is clustering and it consists of two phases. In the first one, nodes are grouped to clusters with the purpose of assigning all nodes of each cluster to the same processor. In the second phase, clusters are merged so as to conform to the number of available processors, then they are mapped to specific processors and finally the nodes of each cluster are scheduled. Examples of such algorithms are Dominant Sequence Clustering (DSC) [9] and Linear Clustering Method (LCM) [10].

Node duplication is a category of task scheduling algorithms where selected tasks are scheduled in more than one processor in an effort to prevent communication costs [11], [12].

Stochastic search algorithms are a broad category of algorithms including Genetic Algorithms (GAs), Simulated Annealing (SA), Particle Swarm Optimization (PSO) and several other algorithms. Such algorithms exploit randomness for steering search to promising regions of the problem space. For the task scheduling problem GAs seems to be the most popular approach [13], [14], [15] reporting very good results. However, the following issues inherent to GAs have to be addressed: need for instance specific parameter calibration, inability to prove optimality and potential long running times. Other stochastic search methods, have also documented cases of promising results for the task scheduling problem [16], [17].

Although heuristics and approximation algorithms are the most common way of tackling the task scheduling problem, mathematical programming can be a competitive alternative and indeed has been proposed for solving the problem [18]. Two major types of mathematical model approaches seem to prevail: sequencing [19] and overlap [20], [21]. In sequencing, specialized variables exist that indicate whether a task is executed after another one at the same processor. In overlap, variables that represent overlapping execution of tasks are incorporated in the model. The overlap model representation seems to scale better than the sequence one since in the former case a significant number of constraints are needed in order to capture the precedence constraints between tasks for each processor. The approach presented in the following section of the paper can be considered as a variation of the overlap model. Although no explicit overlap variables exist, tasks that are not connected in the graph cannot overlap at the same processor due to specialized constraints, while connected tasks are prohibited to execute concurrently due to precedence constraints that exist in the model.

3 Mathematical model (MATH)

The mathematical model assumes two sets of objects. The first one is the set of tasks *T* and the second one is the set of processors *P*. Given that the problem model is heterogeneous each

processor might have different execution time for the same task. The execution time of each task t on each processor p is given by parameter w_{tp}. The communication cost between task t and task t' if they are not scheduled at the same processor are given by parameter $c_{tt'}$. The decision variables of the problem are binary variables y_{tp} defined over each $t \in T$ and $p \in P$ and integer variables x_t defined over each $t \in T$. Each variable y_{tp} assumes value 1 when task t is assigned to processor p and 0 otherwise. On the other hand, each variable x_t assumes the the time value that task t starts execution in the processor dictated by the corresponding y_{tp} variable. Furthermore, since the problem is captured as a DAG consisting from a set of nodes V and a set of edges E it should be noted that the set of tasks and the set of nodes are conceptually identical while the set of edges represents precedence constraints between tasks with weights of edges associated with communication costs.

The objective function of the model is shown in equation (2) representing the target of minimizing the total makespan of the schedule.

$$\text{minimize } x_{t_s} \qquad t_s \text{ is the sink node of the DAG} \tag{2}$$

Three groups of constraints are defined. The first one ensures that each task should be assigned to exactly one processor (3).

$$\sum_{p \in P} y_{tp} = 1 \qquad \forall\, t \in T \tag{3}$$

The second group of constraints states that each task t that is depended on a set of other tasks T' should start execution after all tasks belonging to T' have finished execution. Furthermore, when task t is scheduled to a different processor than task t' that it depends on, the communication cost between them should also be considered. In order to model this constraint three new variables are introduced et_t, $z_{tt'}$ and $k_{ptt'}$. Variable et_t is integer and corresponds to the execution time of task t derived from equation (4). Variable $z_{tt'}$ is binary and assumes value 1 when both tasks t and t' are scheduled in the same processor and 0 otherwise. It gets its value through variable $k_{ptt'}$ which is the product of binary variables $y_{t'p}$ and $y_{t'p}$. Since the product between variables violates the linearity of the model variable $k_{ptt'}$ assumes its value through inequalities (5), (6) and (7). So variable $z_{tt'}$ through equation (8) assumes the value of the sum of the products between variables $y_{t'p}$ and $y_{t'p}$. Finally, inequality (9) states that the difference between execution times for task t and task t' provided that t depends on t' should be no less than the execution time of task t on the designated processor plus the extra communication time needed when tasks are not scheduled in the same processor.

$$et_t = \sum_{p \in P} w_{tp} y_{tp} \qquad \forall t \in T \tag{4}$$

$$k_{ptt'} \leq y_{tp} \qquad \forall p \in P; \forall t, t' \in T: edge_{tt'} \in E \tag{5}$$

$$k_{ptt'} \leq y_{t'p} \qquad \forall p \in P; \forall t, t' \in T: edge_{tt'} \in E \tag{6}$$

$$k_{ptt'} \geq y_{tp} + y_{t'p} - 1 \qquad \forall p \in P; \forall t, t' \in T: edge_{tt'} \in E \tag{7}$$

$$z_{tt'} = \sum_{p \in P} k_{ptt'} \qquad \forall t, t' \in T: edge_{tt'} \in E \tag{8}$$

$$x_t - x_{t'} \geq et_t + c_{tt'}(1 - z_{tt'}) \quad \forall t, t' \in T: edge_{tt'} \in E \tag{9}$$

The third group of constraints guarantees that when two tasks are not connected then they should not overlap if they are to be scheduled in the same processor. Given two tasks *t* and *t'* with no dependencies between them if the start of execution of task *t* (x_t) is less than the start of execution of task *t'* ($x_{t'}$) then x_t plus the execution time of task *t* (et_t) should be less than $x_{t'}$ (10) provided that both tasks execute in the same processor $z_{tt'}$=1. Likewise, when $x_{t'}$ is less than x_t inequality (11) assures that no overlap of execution occurs. Binary variable $m_{tt'}$ ensures that exactly one of inequalities (10) and (11) should hold.

$$(1 - z_{tt'})M + x_{t'} - x_t \geq et_{t'} - M(1 - m_{tt'}) \; \forall t, t' \in T: t, t' \, not \, connected \tag{10}$$

$$(1 - z_{tt'})M + x_t - x_{t'} \geq et_{t'} - M(m_{tt'}) \quad \forall t, t' \in T: t, t' \, not \, connected \tag{11}$$

4 Decomposition of the problem to levels (MATHL)

For DAGs that have more than a few tens of nodes, the model generated becomes too big to be solved using Integer Programming solvers. So, for DAGs of big size the solution is gradually generated scheduling in each step nodes that belong to the same level. Nodes are grouped on levels according to the hops needed for each node to be reached starting from the source node. Initially a sub-problem gets generated that contains only nodes belonging to level 0. A virtual node is temporary added to the graph and connected to all nodes of the 0^{th} level. The start time of this virtual node is the value that should be minimized in this step. Therefore, a mathematical model is constructed including variables that refer on nodes of level 0 only. By solving this model a partial solution is formed that optimally schedules all nodes of this level. Next, nodes of level 1 are scheduled under the preposition that the scheduling of nodes belonging to the previous level cannot change. For level 1 and each subsequent level the mathematical model that will be generated defines variables only for the nodes belonging to the level under consideration while variables that would refer to previous levels have specific values. The procedure continues until the sink node is reached. The procedure described does not guarantee optimal solution to the problem. This is evident from the fact that while solving each level no consideration of nodes belonging to subsequent levels occurs. Nevertheless, satisfactory solutions in acceptable solution time are typically produced since sub-problems solved by the mathematical solver have significant smaller sizes and can be processed in a matter of seconds.

For the rest of the paper the solution process of modeling the full problem and then feeding the model to the Integer Programming solver will be called MATH. On the other hand the solution process of gradually building the schedule using decomposition to levels will be called MATHL.

5 Experiments

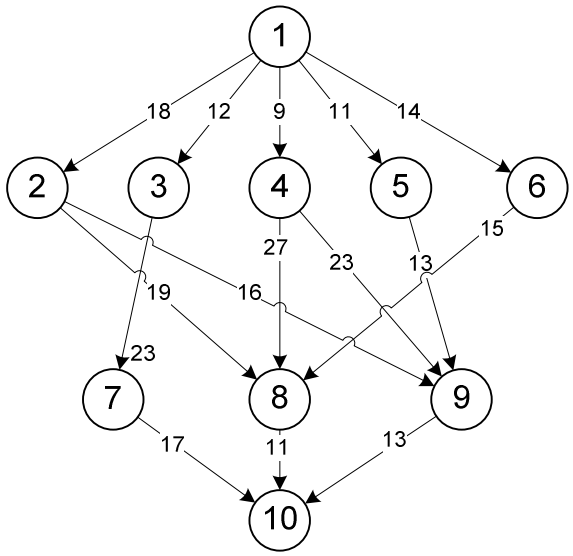
Two categories of experiments were undertaken. The first one used small DAGs up to 15 nodes found in scientific papers in an effort to validate the ability of MATH on constructing optimal schedules by modeling the full problem. The second category used medium to large size DAGs (100 to 1000 nodes) that are generated randomly. In this category the performance of MATHL is analyzed with respect to the results obtained by HEFT and HEFTLA. HEFTLA variation that is used across all experiments is the one that selects for scheduling the task *t*

which minimizes the maximum EFT (Earliest Finish Time) for *t*'s children among all processors where t is tried.

All experiments were performed on a Windows-7 64bit workstation equipped with an Intel Core i7 860 processor and 4GB or RAM. Runs for MATH and MATHL were carried out using GUROBI OPTIMIZER 5.1 Academic as the Integer Programming solver of choice.

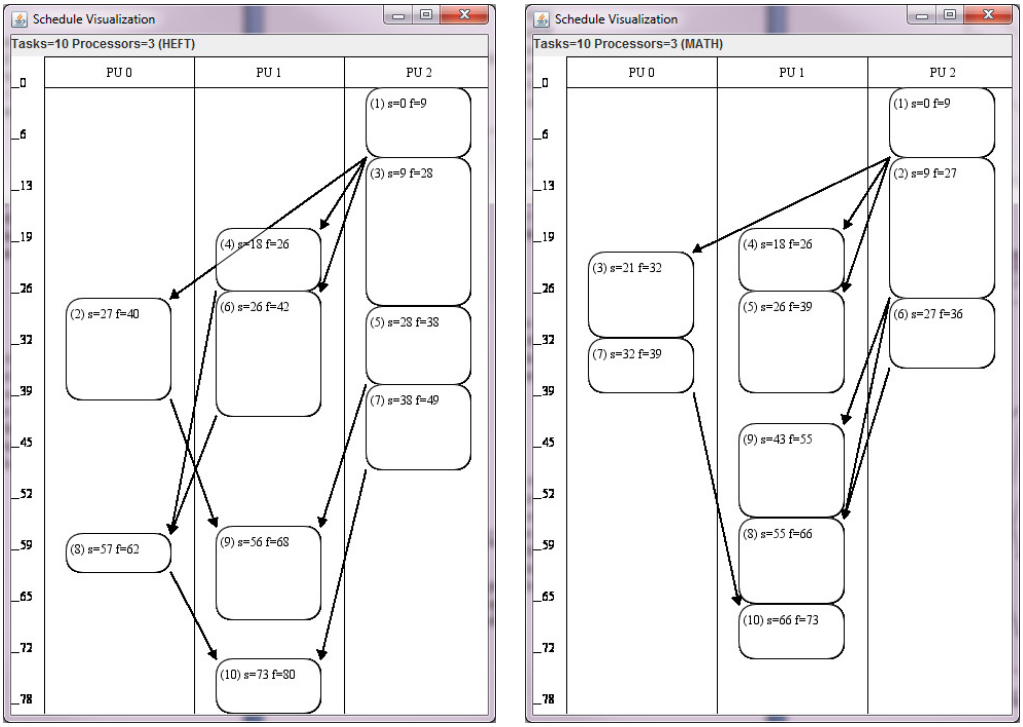
5.1 Experiments on DAGs of small size

An illustrative demonstration of the results obtained by the current approach is given on two small sized sample task graphs taken from the bibliography. Each example DAG is depicted alongside with computation costs of each task in each processor while communication costs are shown in each graph itself. The problem presented in Figure 1 comes from the paper of Topcuoglu et al. [7] (Fig.3 page 265). The authors apply HEFT and CPOP and they manage to achieve schedules with lengths 80 and 86 respectively. MATH algorithm schedules this DAG by achieving the optimal length of 73 time units. The actual schedules resulted from HEFT and MATH are presented in Figure 2.



Task	PU0	PU1	PU2
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

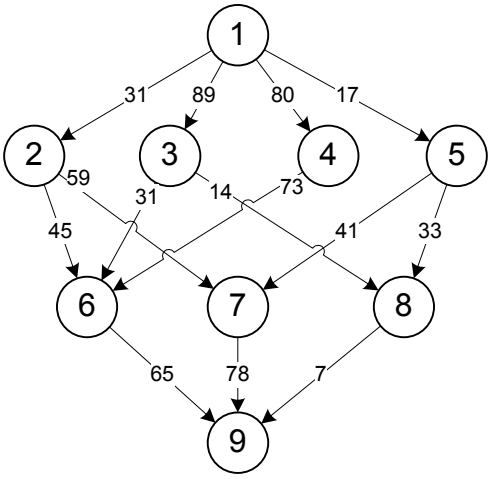
Figure 1. DAG scheduling problem with 10 tasks and 3 processors taken from [7]



(a) (b)

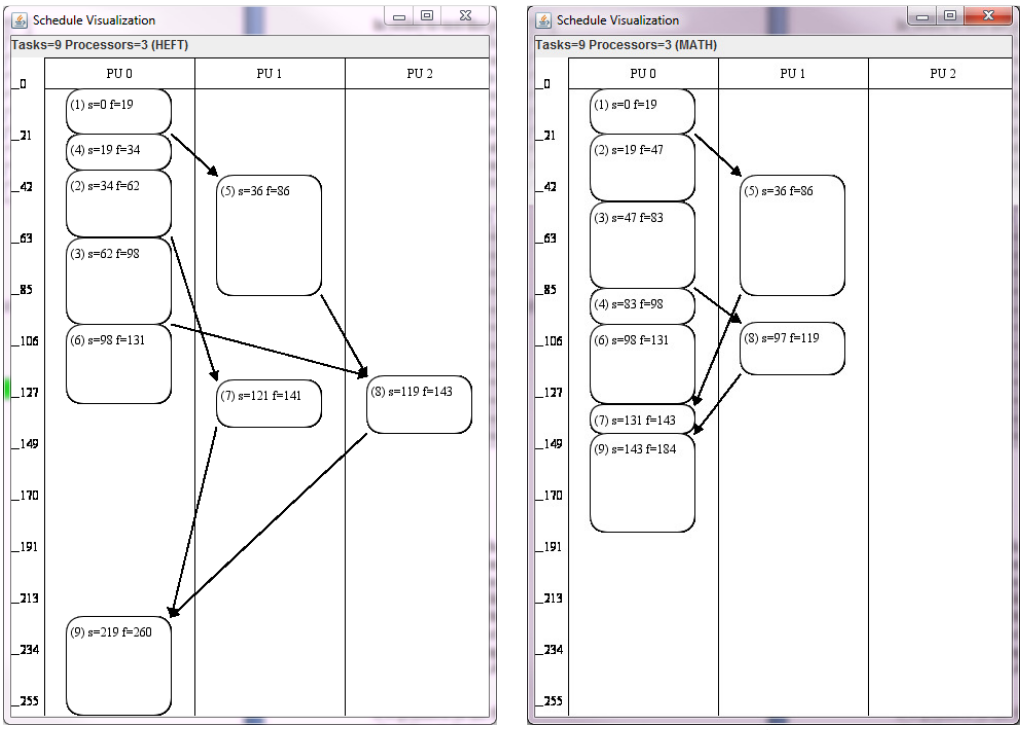
Figure 2. Scheduling of DAG in Figure 1 with (a) HEFT and (b) MATH Solver

The next problem presented in Figure 3 is from the paper of Bittencourt et al. [8] (Fig.1 page 28). Authors apply HEFT and several variations of HEFT that look ahead nodes in the graph before scheduling each node. The schedule length for HEFT is 260 while HEFTLA manages to find the optimal schedule length of 184 as confirmed by the MATH algorithm. Actual schedules resulted from HEFT and HEFTLA, MATH are presented in Figure 4.



Task	PU0	PU1	PU2
1	19	41	34
2	28	46	20
3	36	34	62
4	15	25	37
5	30	50	54
6	33	35	59
7	12	20	21
8	13	22	24
9	41	68	73

Figure 3. DAG scheduling problem with 9 tasks and 3 processors taken from [8]



(a) (b)

Figure 4. Scheduling of DAG in Figure 3 with (a) HEFT and (b) HEFTLA, MATH

Results in Figure 2 and Figure 4 show that MATH finds the optimal results to the above problems. Both results were obtained in time less than a second.

5.2 Experiments on randomly generated DAGs

In order to test the performance of MATHL a number of random graphs were generated. The selected process of random graph generation uses as input the number of tasks and the number of processors. The number of levels that tasks will be grouped in becomes the square root of the number of tasks and nodes are randomly assigned to levels. The execution time of each task on each processor results by multiplication between two random numbers in the range [1,10]. Each node is connected to at least one node and at most every node of the previous level and this is also decided randomly. Edge values representing communication costs are random values produced in the same manner as the node values. In order to normalize the graph total execution time of all tasks on each processor are adjusted so as to become equal to the total communication time of all edges.

Table 1 summarizes the results gathered from experiments performed on randomly generated graphs showing schedule lengths, execution time needed for MATHL and percentage of improvement achieved by MATHL over the best between HEFT and HEFTLA in every case. HEFT seems to produce better results than HEFTLA for almost all cases and MATHL outperforms both of them in all tests. It can be observed that in general the percentage of improvement reduces as the problem gets bigger.

Random Graph	HEFT	HEFTLA	MATHL		% of improvement
100 nodes, 3 processors	*737*	745	**639**	4.66 sec	13.30%
200 nodes, 3 processors	1,115	*1,089*	**997**	37.25sec	8.45%
300 nodes, 3 processors	*1,483*	1,493	**1,310**	41.64 sec	11.67%
400 nodes, 3 processors	*1,869*	1,916	**1,665**	59.53 sec	10.91%
500 nodes, 3 processors	*2,320*	2,357	**2,049**	93.03 sec	11.68%
500 nodes, 4 processors	*1,564*	1,629	**1,498**	190.38 sec	4.22%
500 nodes, 5 processors	*1,298*	1,408	**1,229**	127.69 sec	5.32%
600 nodes, 5 processors	*1,490*	1,553	**1,428**	176.33 sec	4.16%
700 nodes, 5 processors	*1,670*	1,697	**1,603**	238.55 sec	4.01%
800 nodes, 5 processors	*1,832*	1,908	**1,807**	276.62 sec	1.36%
900 nodes, 5 processors	*2,026*	2,125	**2,001**	297.27 sec	1.23%
1000 nodes, 5 processors	*2,216*	2,255	**2,154**	318.16 sec	2.80%

Table 1. Schedule length of HEFT, HEFTLA and MATHL on various random graphs

6 Conclusions

In this paper a mathematical model for the problem of static task scheduling on a system of full connected heterogeneous processors was presented. Preliminary experiments demonstrate that at the expense of more processing time better schedules can be constructed than established heuristic methods like HEFT. The mathematical model proposed can easily solve to optimality small instances of the problem. For bigger problems the solution of decomposing the problem to sub-problems including only tasks belonging to the same level was selected. Results show that for random graphs of up to 1000 nodes the proposed approach consistently outperforms HEFT and HEFTLA.

It is planned in the future to investigate the application of a modified version of the proposed algorithm to heterogeneous systems that are link contention aware.

Acknowledgement: This work is co-funded by the European Union under the 7th Framework Programme under grant agreement ICT-287733**,** project "Architecture oriented paraLlelization for high performance embedded Multicore systems using scilAb (ALMA)".

References

1. J. D.Ullman, NP-complete scheduling problems, Journal of Computing Science, 1975.
2. H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Task scheduling algorithms for heterogeneous processors," in Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth, 1999, pp. 3 –14.
3. T. C. E. Cheng and C. C. S. Sin, "A state-of-the-art review of parallel-machine scheduling research," European Journal of Operational Research, vol. 47, no. 3, pp. 271–292, Aug. 1990.
4. L.-C. Canon, E. Jeannot, R. Sakellariou, and W. Zheng, "Comparative Evaluation Of The Robustness Of DAG Scheduling Heuristics," in Grid Computing, S. Gorlatch, P. Fragopoulou, and T. Priol, Eds. Springer US, 2008, pp. 73–84.
5. O. Sinnen, Task Scheduling for Parallel Systems. John Wiley & Sons, 2007.

6. J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, “Scheduling Precedence Graphs in Systems with Interprocessor Communication Times,” SIAM Journal on Computing, vol. 18, no. 2, pp. 244–257, Apr. 1989.
7. H. Topcuoglu, S. Hariri, and M.-Y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260 –274, Mar. 2002.
8. L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira, “DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm,” in 2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2010, pp. 27 –34.
9. A. Gerasoulis and T. Yang, “A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors,” Journal of Parallel and Distributed Computing, vol. 16, no. 4, pp. 276–291, Dec. 1992.
10. S. J. Kim and J. C. Browne, “A general approach to mapping of parallel computation upon multiprocessor architectures,” in International Conference on Parallel Processing, 1988, vol. 3, p. 8.
11. I. Ahmad and Y.-K. Kwok, “On exploiting task duplication in parallel program scheduling,” IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 9, pp. 872 –892, Sep. 1998.
12. S. Ranaweera and D. P. Agrawal, “A task duplication based scheduling algorithm for heterogeneous systems,” in Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International, 2000, pp. 445 –450.
13. R. C. Correa, A. Ferreira, and P. Rebreyend, “Scheduling multiprocessor tasks with genetic algorithms,” IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 8, pp. 825 –837, Aug. 1999.
14. S. Gupta, G. Agarwal, and V. Kumar, “Task Scheduling in Multiprocessor System Using Genetic Algorithm,” in 2010 Second International Conference on Machine Learning and Computing (ICMLC), 2010, pp. 267 –271.
15. O. L. Sathappan, P. Chitra, P. Venkatesh, and M. Prabhu, “Modified genetic algorithm for multiobjective task scheduling on heterogeneous computing system,” International Journal of Information Technology, Communications and Convergence, vol. 1, no. 2, pp. 146–158, Jan. 2011.
16. T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems,” Journal of Parallel and Distributed Computing, vol. 61, no. 6, pp. 810–837, Jun. 2001.
17. P.-Y. Yin, S.-S. Yu, P.-P. Wang, and Y.-T. Wang, “A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems,” Computer Standards & Interfaces, vol. 28, no. 4, pp. 441–450, Apr. 2006.
18. A. Davare, J. Chong, Q. Zhu, D. M. Densmore, and A. L. Sangiovanni-Vincentelli, “Classification, Customization, and Characterization: Using MILP for Task Allocation and Scheduling,” Technical Report Identifier: EECS-2006-166, EECS Department, University of California, University of California at Berkeley, California, Dec. 2006.
19. P.E. Coll, C.E. Ribeiro, and C.C. de Souza. “Multiprocessor scheduling under precedence constraints: polyhedral results.” Discrete Applied Mathematics, vol 154, no. 5, pp. 770-801, 2006.
20. S. Prakash and A. C. Parker, “SOS: Synthesis of application-specific heterogeneous multiprocessor systems,” Journal of Parallel and Distributed Computing, vol. 16, no. 4, pp. 338–351, Dec. 1992.
21. M.F.Tompkins. “Optimization techniques for task allocation and scheduling in distributed multi-agent operations.” Master’s thesis, MIT, June 2003.

MISTA 2013

A Reinforcement Learning Approach to Solving Hybrid Flexible Flowline Scheduling Problems

Bert Van Vreckem · Dmitriy Borodin · Wim De Bruyn · Ann Nowé

Abstract In this paper, we present a method based on Learning Automata to solve Hybrid Flexible Flowline Scheduling Problems (HFFSP) with additional constraints like sequence dependent setup times, precedence relations between jobs and machine eligibility. This category of production scheduling problems is noteworthy because it involves several types of constraints that occur in complex real-life production scheduling problems like those in process industry and batch production. In the proposed technique, Learning Automata play a dispersion game to determine the order of jobs to be processed in a way that makespan is minimized, and precedence constraint violations are avoided. Experiments on a set of benchmark problems indicate that this method can yield better results than the ones known until now.

1 Introduction

Within the vast research field of scheduling, the optimisation of production processes is an area that has known considerable progress in the last years. Originally, the only thing computers could solve to optimality, or even to a feasible solution, were toy problems. In some industries, where the production process is not a very complex one, automated scheduling solutions have been in use for some time. On the other hand, in industries like the process industry, where very hard constraints are imposed on production schedules, the planning and scheduling is still mainly a manual job, done

Bert Van Vreckem
University College Ghent
E-mail: bert.vanvreckem@hogent.be

Dmitriy Borodin
OM Partners
E-mail: dborodin@ompartners.com

Wim De Bruyn
University College Ghent
E-mail: wim.debruyn@hogent.be

Ann Nowé
Vrije Universiteit Brussel
E-mail: ann.nowe@vub.ac.be

by highly experienced people that know the production line inside out. Recent advances in computer hardware however (more specifically faster processors and algorithms) have made it ossible for production scheduling problems of more realistic size to be tackled.

The typical approach in the operational research community is to model the scheduling problem as a Mixed Integer (Linear) Programming model (MIP/MILP) (Mendez et al, 2006) and then solve it to optimality with methods that are suitable for this formalisation. Nevertheless, this method still has its limits. Setting up a MILP model is hard, and involves a particular expertise. The way constraints are formulated can have a serious impact on the efficiency of the solver. Additionally, this model is rarely reusable for other, even similar production processes and is hard to adapt to changes in the plant. Furthermore, in order to make the problem tractable, often simplifications are introduced that no longer reflect reality (e.g. only allowing to allocate resources for blocks of four hours, which would be useless in a real-life setting). This greatly decreases the usefulness of the solutions found (Gicquel et al, 2012).

Because of the above reasons, heuristics and meta-heuristics are still being considered as an interesting alternative to exact methods like MILP, as they are able to produce solutions of high quality in a reasonable time. The main disadvantage is that they no longer guarantee that found solutions are optimal, and it is generally not possible to estimate how far they are from an optimal solution. However, plant managers are usually neither looking for the optimum at all (especially if that solution turns out to be unusable in practice), nor for a completely automated black box scheduling algorithm. In many cases, a "better than manual" solution would already be a great improvement over current practice.

One approach is to start with a MILP formulation and apply techniques from the class of heuristics or meta-heuristics. The advantages include relatively fast response time, tractability of the problem, scalability while it is still required to define a MILP model and incorporate an appropriate heuristic within the defined MILP. An overview is provided in the survey by Ribas et al (2010). For a real-life case-study of applying genetic algorithm "on top" of a MILP model we refer to Borodin et al (2011).

2 Hybrid Flexible Flowline Scheduling Problems

In this paper, we introduce a learning approach, and apply it to Hybrid Flexible Flow line Scheduling Problems (HFFSP) with a set of additional restrictions (Ruiz et al, 2008). This is a particularly interesting type of scheduling problems, as it involves several hard constraints that occur in complex real-life production processes. In a Flow line Scheduling Problem (FSP), a number of jobs have to be processed in a set of consecutive stages. In each stage, there are a number of unrelated parallel machines that can process jobs (this is the *hybrid* case of FSP). Jobs are processed at each stage in the same order and by exactly one machine. Some stages can be skipped (this is the *flexible* case of FSP) and for some jobs, not all machines are eligible to process it. Jobs may have precedence relations. There may be a time lag between the processing of a job in two consecutive stages that can be either positive or negative. Machine-based sequence-dependent setup times are also considered.

In his PhD thesis, Urlings (2010) explores this type of scheduling problem in detail. First, a MIP model for HFFSPs is introduced that is applied to a set of benchmark problems first published in (Ruiz et al, 2008). Subsequent chapters of the thesis discuss the solution strategies based on CPLEX, heuristics, genetic algorithms, and local search

techniques. Even in smaller problems with five jobs, given the extra real-life constraints, CPLEX was unable to find an optimal solution for all instances. In larger problems, it found the optimum in only a fraction of the cases. For example, in a subset of problem instances with 15 jobs, two stages and one machine per stage (which does not seem to be an overly large problem), only in 0.26% of the cases an optimal solution was found. A statistical analysis of the MIP model revealed that the difficulty of the problem instances is not caused by the number of jobs, but primarily by the precedence relations between jobs.

A typical approach for solving an FSP is to first determine the order in which the jobs are processed, i.e. a job permutation. In the next phase, machines in each production stage are assigned to jobs, and start times are determined according to machine availability. The job permutation and machine assignments will determine the quality of the schedule, e.g. the makespan. Heuristics or machine learning techniques can be used in either or both stages. For example, Kurz and Askin (2004) research the application of several different methods to flexible flow line scheduling problems with sequence-dependent setup times, namely greedy heuristic, flow line methods, Insertion Heuristic and Random Keys Genetic Algorithm (RKGA). Each of these heuristics assumes the job ordering first, and then assignment of jobs to machines. RKGA was found to be superior in the majority of cases in terms of solution quality and solution time. Tavakkoli-Moghaddam et al (2009) introduced a method where the job permutation is selected by tabu search and then the assignment/timing procedure uses a number of dispatching rules to determine the detailed schedule and resulting makespan. Zandieh et al (2009) proposed a robust genetic algorithm for solving real flow line scheduling problems. The solution representation of the genetic algorithm itself comprised both job sequencing and machine assignment parts, though a set of underlying heuristics was used in the sense of machine assignment rules. A bit different approach was demonstrated by Borodin et al (2011) where the authors first performed the job assignment to machines using genetic algorithm, and then determined the job order. The approach was based on the MILP formulation of the problem: once the job assignment to machines had been fixed, the original MILP resulted in an easy-to-solve LP, and its solution allowed to construct a resulting schedule.

This paper contributes to the research domain by introducing a Reinforcement Learning (RL) approach based on Learning Automata (LA) for solving complex real-life HFFSPs. Rather than starting with a MIP formulation of the problem, we used a straightforward model that is easily implemented in a mainstream programming language, without trying to represent constraints in a way that improves efficiency of a solver. We use Learning Automata to find job permutations that lead to high quality schedules. We currently do not apply learning in the machine assignment phase, but we use a heuristic that performed well in previous experiments (Urlings, 2010).

The rest of the paper comprises the following sections. In the Learning Automata section RL and LA are introduced, after which we describe the proposed algorithm in detail. Then, computational results are presented and compared to the best known ones for the set of benchmarks in Experiments. Finally, we discuss conclusions and propose ideas for future research.

3 Learning Automata

Reinforcement Learning (RL) is a technique that allows an agent to learn how to maximize a numerical reward signal. A learning agent must determine the actions that yield the most reward by a trial-and-error approach, in contrast to supervised learning where the best action would be known a priori. In real-life cases, the selected actions affect not only the immediate reward but also the reward of the next situation and, consequently, all following rewards. The trial-and-error search and delayed reward are the two most important distinguishing features of RL (Sutton and Barto, 1998).

In a RL setting, an agent percieves the state of its environment at instant t and based on this, chooses an action $a(t)$ from a specific set $A = a_1, \ldots, a_n$. This action may change the state of the environment and generates a reinforcement signal $r(t)$ that is received by the agent. The learning task consists of finding a strategy that maximizes long term cumulative reward.

Learning Automata (LA) (Narendra and Thathachar, 1974) are RL agents that determine their actions according to a probability distribution $p(t) = p_1, \ldots, p_n$, with $p_i = prob[a(t) = a_i]$, and with the condition that $\sum_{i=1}^{n} p_i = 1$. Initially, all actions are chosen according to the same probability $p_i = \frac{1}{n}$, and these are updated using the reinforcement signal from the environment, usually 0 or 1 for a penalty, resp. reward signal. The general update rule for Learning Automata is formulated as follows:

$$\begin{aligned} p_i(t+1) = p_i(t) \; & +\alpha_{rew} r(t)(1 - p_i(t)) \\ & -\alpha_{pen}(1 - r(t))p_i(t) \\ & \text{if } a_i \text{ is the action taken at instant } t \end{aligned} \tag{1}$$

$$\begin{aligned} p_j(t+1) = p_j(t) \; & -\alpha_{rew} r(t) p_j(t) \\ & +\alpha_{pen}(1 - r(t)) \left(\frac{1}{n-1} - p_j(t) \right) \\ & \text{if } a_j \neq a_i \end{aligned} \tag{2}$$

with α_{rew} and $\alpha_{pen} \in [0, 1]$ the reward and penalty rate. If $\alpha_{rew} = \alpha_{pen}$, the update rule is called linear reward-penalty (L_{R-P}), when $\alpha_{pen} = 0$, it is referred to as linear reward-inaction (L_{R-I}). There are more variations to this update scheme, but those are beyond the scope of this paper. In a stationary environment, these update schemes can be proven to converge to an optimal strategy (Narendra and Thathachar, 1974).

4 Learning permutations with precedence constraints

Wauters (2012) proposes a method called Probabilistic Basic Simple Strategy (PBSS) that uses Learning Automata to learn a permutation, and he applies it a.o. to solve complex Resource Constrained Multi-Project Scheduling Problems (RCMPSP). Every position in the permutation is assigned a Learning Automaton that chooses the job for that position in the permutation, using its probability distribution. Every time instant t, the LAs play a *dispersion game*, i.e. each has to choose a unique job. Initially, since action probabilities are equal, some LAs will choose the same job. If such collisions occur, they will have to try again. Those LAs that did choose a unique job, will stick to their choice, or differently put, the action probability for that job is set to 1, and for

all other jobs to 0. In addition, other automatons will be forbidden to choose that job for the following runs of the dispersion game, i.e. action probabilities for that job in those other LAs is set to 0. For each agent, the probabilities that are still non-zero are then renormalized so that their sum equals 1. Then, the dispersion game is played again, until all LAs have chosen a unique job. The resulting permutation is then evaluated according to some quality function. If the result is better than the best one found thus far, $r(t) = 1$, if not, $r(t) = 0$. This signal is then used to update the probabilities from the *beginning* of the time instant, before the dispersion game started, according to L_{R-I}. In the next time instant $t+1$, the LAs play the dispersion game again with the updated probabilities. After a number of iterations, the action probabilities of the LAs converge to pure policies where one specific, unique, action is chosen that, combined, yield a permutation of high quality. This method can be used, and performs extremely well, in many situations where learning permutations is involved.

Unfortunately, PBSS can not be readily applied to the HFFSP. In this class of scheduling problems, precedence constraints between jobs may severely limit the number of valid permutations. Given the update scheme proposed in PBSS, agents don't learn to avoid violations of precedence constraints. This is a consequence of the L_{R-I} update scheme that only learns from positive feedback. Indeed, in preliminary experiments on instances with precedence constraints, the Learning Automata never came up with a valid schedule, therefore no learning happened at all. In this paper, we propose a variation of PBSS where generating invalid schedules also results in a learning experience:

- If the job permutation is valid, perform a L_{R-I} update in all agents, depending on the resulting makespan ms and best makespan until now ms_{best}:
  - improved: $r(t) = 1$;
  - equally good: $r(t) = 0.5$;
  - worse: $r(t) = ms_{best}/ms/2$;
  - no valid schedule found: $r(t) = 0$;
- If the job permutation is *invalid*, perform a L_{R-P} update with $r(t) = 0$ for all agents that are involved in the violation of precedence constraints.

The main difference from PBSS is that we allow to learn explicitly from negative experience by extending the L_{R-I} update with an L_{R-P} component.

In the following section, we present some results of experiments on HFFSP instances with this update scheme.

5 Experiments

We used the problem set from Ruiz et al (2008)[1] to evaluate the performance of Learning Automata for solving HFFSPs. The set is subdivided according to the number of jobs, respectively 5, 7, 9, 11, 13, and 15. For each subset, there are 96 instances with other specific characteristics (e.g. precedence constraints or not, skipping stages or not, ...).

We used 0.1 for the reward α_{rew} rate and 0.5 for the penalty rate α_{pen}. We did not perform careful parameter tuning, but these values seem to perform well. The algorithm was run until the LAs converged to a pure strategy, where each agent chooses a unique

[1] Available at `http://soa.iti.es/problem-instances`

Instance set	5	7	9	11	13	15	overall
mean RD (%)	0.0697	2.0131	1.1568	1.6565	3.7294	7.9189	2.7484
best RD (%)	-35.70	-24.71	-26.92	-21.10	-43.34	-10.46	-43.34
instances improved	11	12	18	12	9	6	68
instances equal	62	40	19	18	8	7	154
instances worse	23	44	59	66	79	82	354

Table 1 Summary of results. Columns denote the instance sets, according to the number of jobs. The top row are mean relative deviations (in percentages) w.r.t. the best known values. Next, best relative deviations for each set are given (also in percentages). Negative values are improvements over the best known result. The three bottom rows denote the number of instances where we got respectively improvements, the same, and worse results w.r.t. best known values.

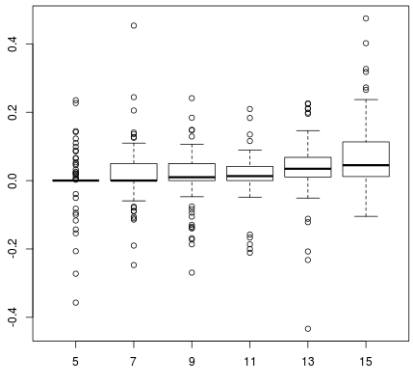


Fig. 1 Box plots for the relative deviations of makespan (in percent) from the best known results. Negative values indicate an improvement. One may expect that results will be worse as the number of jobs increases, but this is evidently not the case in the set of instances with 13 jobs that yields better results than the ones with 11 jobs. Remark that the number of jobs is *not* the only indicator of the difficulty of the problem instance. For example, precedence constraints between jobs may make the problem much more complex.

job every time, or for at most thirty seconds after which the best found schedule so far is selected as the final solution. We took a fixed machine assignment rule from Urlings (2010), called Earliest Preparation Next Stage (EPNS) as it performs well overall in his experiments.

For each instance, the schedule with the shortest makespan that was encountered during a run was chosen as the result. In all subsets, we obtained schedules with a shorter makespan than the best known until now for some of the instances. Table 1 contains a summary of the results. In Figure 1, we show box plots for the relative deviations with regard to the best known makespans in each instance subset. Finally, Figure 2 shows an example of a schedule where substantial improvement was achieved.

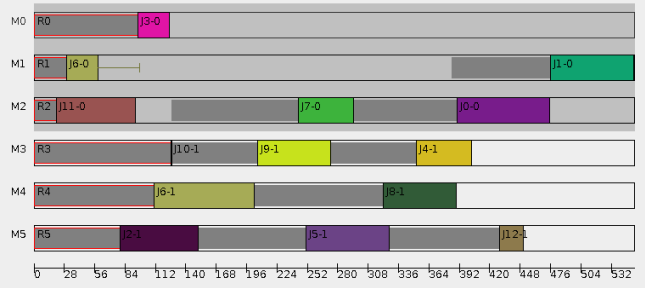


Fig. 2 The resulting schedule for an instance with 13 jobs, viz. "Ism_13_2_3_1-200_50_50_75-125_50-100_-99-99_1-3_3". The resulting makespan (553) is an improvement of 43% over the reported best known (976).

6 Discussion

In this paper, we introduced a technique based on Learning Automata to solve highly constrained Hybrid Flexible Flowshop Scheduling Problems. These are not made difficult by the problem size (specifically, the number of jobs), but by other constraints that occur in real-life scheduling problems, in particular precedence relations. We started from a simple, natural model of the problem without trying to represent constraints in a way that could make the algorithm more efficient. We presented the results of experiments on a set of benchmark problems and showed that we could improve on the best known solutions in several cases. By tuning the experimental parameters, such as the reward and penalty rate, we should be able to get even better results.

At this stage, we only focused on determining the order in which jobs are scheduled, but did not look at machine assignments in great detail. This area should also provide opportunities for improving results.

It is interesting to note that the L_{R-P} update rule is considered to have weaker convergence guarantees than L_{R-I} (Narendra and Thathachar, 1974). However, in this setting with precedence constraints, there are only few or no positive learning experiences. The Learning Automata would often select an invalid job sequence, and are unable to avoid them. The L_{R-P} component of our solution strategy appears to help achieve convergence more quickly. On larger instances with precedence constraints, the algorithm has more difficulties to converge, so further investigation into its behaviour is definitely needed.

References

Borodin D, Van Vreckem B, De Bruyn W (2011) Case study: an effective genetic algorithm for the chemical batch production scheduling. In: Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA 2011)

Gicquel C, Hege L, Minoux M, van Canneyt W (2012) A discrete time exact solution approach for a complex hybrid flow-shop scheduling problem with limited-wait constraints. Computers & Operations Research 39(3):629–636

Kurz M, Askin R (2004) Scheduling flexible flow lines with sequence-dependent setup times. European Journal of Operational Research 159(1):66–82

Mendez C, Cerda J, Grossmann I, Harjunkoski I, Fahl M (2006) State-of-the-art review of optimization methods for short-term scheduling of batch processes. Computers & Chemical Engineering 30(6-7):913–946

Narendra KS, Thathachar MAL (1974) Learning automata–A survey. IEEE Transactions on Systems, Man, and Cybernetics SMC-4(4):323–334

Ribas I, Leisten R, Framiñan J (2010) Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers & Operations Research 37(8):1439–1454

Ruiz R, Şerifoğlu F, Urlings T (2008) Modeling realistic hybrid flexible flowshop scheduling problems. Computers & Operations Research 35(4):1151–1175

Sutton RS, Barto AG (1998) Reinforcement Learning: An Introduction. The MIT Press

Tavakkoli-Moghaddam R, Safaei N, Sassani F (2009) A memetic algorithm for the flexible flow line scheduling problem with processor blocking. Computers & Operations Research 36(2):402–414

Urlings T (2010) Heuristics and metaheuristics for heavily constrained hybrid flowshop problems. Phd thesis, Universidad Politécnica de Valencia

Wauters T (2012) Reinforcement learning enhanced heuristic search for combinatorial optimization. Phd thesis, KULeuven

Zandieh M, Mozaffari E, Gholami M (2009) A robust genetic algorithm for scheduling realistic hybrid flexible flow line problems. Journal of Intelligent Manufacturing 21(6):731–743

MISTA 2013

A Robust Multi-Agent Reinforcement Learning Approach for Scheduling Problems

Yailen Martínez · Ann Nowé · Rafael Bello

Abstract Scheduling is a decision making process that is used on a regular basis in many real life situations, for example, manufacturing scheduling takes care of the allocation of limited manufacturing resources over time among parallel and sequential manufacturing activities. The vast majority of the research in scheduling has focused on the development of algorithms assuming complete information and a deterministic environment. However, the real world is not so stable, projects may be subject to unexpected events during execution, which may lead to numerous schedule disruptions, for example, resources can become unavailable, new orders can arrive, operations could take longer than expected, that is why it is so important to develop algorithms able to deal with such disruptions without having to reschedule. In this paper, different types of approaches to deal with uncertainty are studied. Based on this, a new approach to incorporate robustness in the schedule construction process is proposed. Some experiments are developed in order to show how the new approach works and how it compares to the existing techniques.

1 Introduction

Scheduling is a very active research field with a high practical relevance. It takes care of the allocation of resources to tasks over time, and its goal is to optimize one or more objectives. Depending on the problem being solved, tasks and resources can take many forms, and the objectives can also vary [14]. In this paper we are interested in manufacturing scheduling, which is an optimization process that allocates limited

Yailen Martínez
Vrije Universiteit Brussel
Universidad Central "Marta Abreu" de Las Villas
E-mail: ymartine@vub.ac.be, yailenm@uclv.edu.cu

Ann Nowé
Vrije Universiteit Brussel
E-mail: ann.nowe@vub.ac.be

Rafael Bello
Universidad Central "Marta Abreu" de Las Villas
E-mail: rbellop@uclv.edu.cu

manufacturing resources over time among parallel and sequential manufacturing activities. Customer orders have to be executed, and each order is composed by a number of operations that have to be processed on the resources or machines available. Each order can have release and due dates associated, and typical objectives functions involve minimizing for example, the tardiness or the makespan. The vast majority of the research in scheduling has focused on the development of algorithms assuming complete information and a deterministic environment. However, the real world is not so stable, projects may be subject to unexpected events during execution, which may lead to numerous schedule disruptions, for example, resources can become unavailable (breakdowns or scheduled maintenances), new orders can arrive or operations could take longer than expected. That is the reason why companies prefer to have robust schedules rather than optimal ones. A key issue is to find a proper balance between these two performance measures.

One approach to deal with such disruptions is to generate robust schedules, where robustness refers to the performance of an algorithm under uncertainties. The objective is to generate schedules that are able to absorb some level of uncertainty without having to reschedule. There are some techniques that can deal with such problems, in this paper we will consider slack-based techniques, where the main idea is to provide each activity with some extra execution time (slack time) in such a way that part of the uncertainty can be absorbed.

Scheduling problems can be interpreted as a distributed sequential decision making task, which makes the use of multi-agent reinforcement learning algorithms possible. These algorithms are of high relevance to various real-world problems, as they allow the agents to learn good overall solutions through repeated interactions with the environment. Learning approaches have been used in the solution of different scheduling problems. According to the literature, interpreting them as multi-agent learning problems is beneficial for obtaining near-optimal solutions. In this paper, a new way to increase the robustness in the solutions obtained by a Reinforcement Learning approach is proposed. The experiments show that the approach is able to obtain near-optimal solutions, and it gives the possibility of increasing the level of robustness by playing with different parameters defined along the paper. It is important to mention that the level of robustness can be increased without loosing too much in optimality.

2 Reinforcement Learning and Multi-Agent Systems

Reinforcement Learning (RL) is learning what to do (how to map situations to actions) so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the highest reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics, trial-and-error search and delayed reward, are the two most important distinguishing features of Reinforcement Learning [16].

In the standard RL model, an agent is connected to its environment via perception and action. In each interaction step, the agent perceives the current state s of its environment, and then selects an action a to change this state. This transition generates a reinforcement signal r, which is received by the agent. The task of the agent is to learn a policy for choosing actions in each state to receive the maximal long-run cumulative

reward. Reinforcement Learning methods explore the environment over time to come up with a desired policy [17].

At each time t, the agent perceives its state $s_t \in S$ and the set of possible actions $A(s_t)$. It chooses an action $a \in A(s_t)$ and receives from the environment the new state s_{t+1} and a reward r_{t+1}, this means that the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π_t, where $\pi_t(s, a)$ is the probability that $a_t = a$ if $s_t = s$. In words, it is the probability of selecting action a in state s at time t.

One of the challenges that arise in RL is the trade-off between exploration and exploitation. To obtain a high reward, a RL agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. Therefore the agent must sample the available actions sufficiently and progressively favor those that appear to be best.

A well-known RL algorithm is Q-Learning (QL), which is based on learning an action-value function that gives the expected utility of taking a given action in a given state. The core of the algorithm is a simple value iteration update, each pair (s, a) has a Q-value associated. When the action a is selected by the agent located in state s, the Q-value for that state-action pair is updated based on the immediate reward received when selecting that action, and the best Q-value for the subsequent state s'. The update rule for the state action pair (s, a) is the following:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \tag{1}$$

In this expression, $\alpha \in \{0, 1\}$ is the learning rate and r the reward or penalty resulting from taking action a in state s. The learning rate α determines the degree by which the old value is updated. For example, if the learning rate is 0, then nothing is updated at all. If, on the other hand, $\alpha = 1$, then the old value is replaced by the new estimate. Usually a small value is chosen for the learning rate, for example, $\alpha = 0.1$. The discount factor (parameter γ) has a range value of 0 to 1 ($\gamma \in \{0, 1\}$). If γ is closer to zero, the agent will tend to consider only immediate reward. If γ is closer to one, the agent will consider future reward with greater weight.

Learning approaches have been used in the solution of different scheduling problems, for example in [17], where the authors focus on the application domain of space shuttle payload processing (SSPP) for NASA. In [5] and [6], the authors propose the application of reinforcement learning techniques in the solution of the job shop scheduling. The results obtained show that interpreting it as a multi-agent learning problem is beneficial for obtaining near-optimal solutions, which can compete with alternative solution approaches. This idea was extended in [12] and [13], where the flexible job shop scheduling and also an online scheduling scenario are modeled and solved using RL techniques. In [11], a generic multi-agent reinforcement learning approach is proposed, which can be adapted to solve different scheduling scenarios. In the next section we will briefly explain the main idea behind this approach, as it is here where we propose to include some modifications in order to be able to obtain near-optimal solutions, but also giving the possibility to deal with unexpected events, without loosing too much in optimality.

2.1 Multi-Agent Reinforcement Learning approach for scheduling problems

When solving a scheduling problem, for example the Job Shop Scheduling Problem (JSSP), the learning algorithm takes care of the sequencing of the operations on the machines. It receives the assignments of machines to operations and decides in which order to execute them on the different resources. In order to do this, we have an agent per machine choosing which operation to process next from the queue of operations waiting at the corresponding resource. To start the algorithm, every job releases its first operation at time 0, all these operations go to the machine they have assigned and start to be processed, if two or more operations go to the same machine then only one of them is selected and the rest remain in the queue until the machine is available again.

To choose an action under the Q-Learning algorithm, the agent takes into account the Q-values associated to the possible operations to execute at that time step in the corresponding machine. In order to balance exploration and exploitation, an epsilon greedy policy is used. This means that the agent has a small probability of selecting an action at random, and a higher probability of selecting the best action, in this case the operation with the highest Q-Value associated. This leads to the definition of the feedback signals the agent will receive.

There are different possible feedback signals that can be used when solving a scheduling problem. For example, in [7], the authors use a cost function as feedback to the agent, this cost is defined as the number of operations present, at that point in time, in the queues of the whole system. This means that high costs are incurred when there are many operations waiting for further processing in the system and, hence, the overall utilization of the resources is poor. Note that we are using costs as reward signals, meaning that the lower the cost the better the action.

We perform some experiments with this cost function and also define other two functions. The first one uses the processing times of the operations as feedback signal. In this case, the immediate reward (cost in this case) is the processing time of the operation that is selected to be processed next, based on the idea of the Shortest Processing Time (SPT) dispatching rule. The second one is based on the cost function proposed in [7], but instead of using the number of operations in the queues of the whole system, we use the sum of the expected processing times of these queued operations. At some point during the scheduling process the length of the queues can be the same, but the sum of the processing times of the operations can be very different. Hence, this function better represents the overall state of the system. The experiments show that the idea of using the processing time of the operations as feedback signal is able to yield the best results. For a more detailed description of the learning algorithm and how it can be adapted to different scheduling scenarios we refer to [11]. In the next sections we describe how to increase the robustness in the solutions being obtained by the approach.

3 Slack-based techniques

During the scheduling process machines can break down, operations can take longer than expected, new orders can arrive, the priority of some orders might change, and all these disruptions will eventually make the original schedule fail. All these unexpected

events can cause delays in the delivery of the products (due dates might be violated), increasing the idle times of the machines and the staff, among many other consequences.

Therefore, in real environments it may not be useful to spend significant efforts to produce optimal solutions, since the optimality is achieved only if the solution can be executed as planned. On the contrary, as noted in [15], *"a sub-optimal schedule that contains some built-in flexibility for dealing with unforeseen events, might provide useful characteristics for the execution"*. In the search of optimal solutions, machines are constantly working and the operations are executed as soon as possible in order to meet the due dates and minimize objectives like the makespan or the tardiness. That is why most of the time we are faced with non-delay schedules, which are good in practice as long as unexpected events do not occur. It is important to say that even though the presence of idle times is usually considered as bad when scheduling, it is good in performance, because it is what keeps delays from propagating.

One way to deal with possible disruptions is by generating robust schedules [4], that is, schedules with the ability to satisfy performance requirements predictably in an uncertain environment [10]. In many decision processes it is common that solutions with a certain level of robustness are required in order to maintain their feasibility in settings with incomplete or imprecise data. There are different approaches that can be used in order to deal with uncertainty and robustness in scheduling, either finding solutions which adapt dynamically to changes or incorporating available knowledge about possible changes to the solution [9]. This leads to two possible classifications: *proactive approaches* (also called predictive) or *reactive approaches*.

The goal of proactive scheduling is to take into account possible disruptions while constructing the original schedule. This allows to make the predictive schedule more robust, as it has some statistical knowledge of uncertainty [3]. Proactive scheduling has also been defined as techniques that seek to produce an off-line schedule that is robust to execution time events [1]. The utility of this type of approach depends on whether the uncertainty can be quantified in some way (like knowing the mean time between failure for the machines etc). If the information is available, it can be used by proactive techniques. If however, the degree of uncertainty is very high, a more reactive approach is needed. In this paper we focus on proactive techniques.

In order to obtain robust schedules, an intuitive approach consists in adding some form of redundancy to the solution during the scheduling process. Figure 1 shows an example of two different solutions for the same scheduling problem.

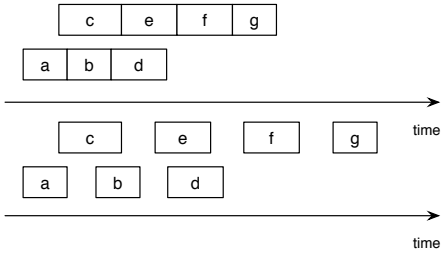


Fig. 1 Increasing robustness by adding redundancy. Schedule b) is considered more robust than schedule a).

Schedule a) is the classical example of a non-delay schedule where the objective is to optimize the resource utilization, that is why each activity starts as soon as its predecessor ends. On the other hand, schedule b) presents some idle intervals between the execution of the different activities. For example, activity 'b' does not start immediately after activity 'a' is finished. This approach allows to tolerate some changes in the original problem like delay in the activity processing and machine breakdowns. At the same time, by introducing redundancy the resource utilization is reduced, which means that objective functions like minimizing the makespan or the tardiness will also be affected. Therefore, the main goal of the redundancy-based approaches is to find out how much redundancy is needed and how it has to be distributed over the whole solution. These techniques which base their behavior in adding redundancy will be referred to as slack-based techniques and are explained in detail in the next section.

4 Slack-based techniques

According to the literature, the main idea behind slack-based techniques is to provide each activity with some extra execution time so that some level of uncertainty can be absorbed without rescheduling. In order to do that, these techniques base their behavior on existing knowledge about failures of the machines, for example: the mean time between failures, or the mean down time of a resource. In this section we describe three slack-based techniques: Temporal Protection, Time Windows Slack and Focused Time Windows Slack. Temporal Protection, first proposed by Chiang and Fox [2], is a technique that is based on the idea of predictively building a schedule taking into account previous knowledge of uncertainty. Resources that have a non-zero probability of breakdown are identified as *breakable* resources. The duration of all the activities requiring breakable resources is extended in order to provide extra time to cope with the breakdowns and the scheduling problem with protected durations is then solved with standard scheduling techniques [8].

One important point when using the temporal protection, or any other slack-based technique is to decide the amount of extra time that will be added to each activity. Too much protection will end up with a poor quality schedule, but a highly robust one. Too little protection will result in a poor quality schedule execution if a breakdown occurs. The approach presented in [8] extends the duration of each activity taking into account the expected time between machine failures and the duration of the breakdown or interruption. It is important to remember that if breakdowns do not occur, or if they do occur, but take less time than the slack time added, operations that are scheduled after a 'protected' operation can start their processing earlier. If we analyze the way the Temporal Protection technique uses to add the slack times, we will notice that there are situations where it will not be possible to 'share' slack between activities, even if there are no breakdowns.

In order to avoid such situations, where the slack time added can not be used by upcoming operations, the Time Window Slack (TWS) approach was proposed in [4]. In this approach the authors change the way of adding slack to the operations in order to make sure that the schedule has sufficient slack time for each activity. It is important to mention that the required slack for an activity under TWS is considerably longer than the duration extension in temporal protection. More specifically, the amount of slack on each activity is equal to the sum of the durations of all the expected breakdowns of resource R, and this is mainly due to the fact that authors expect the slack of

all the activities on each resource to be shared [4]. The slack-times in this approach are also calculated based on the mean time between failures on the resources and the breakdown durations.

Figure 2 (left-hand side) shows how the TWS method adds extra execution time on every operation, assuming that the slack of all the activities on each resource will be shared.

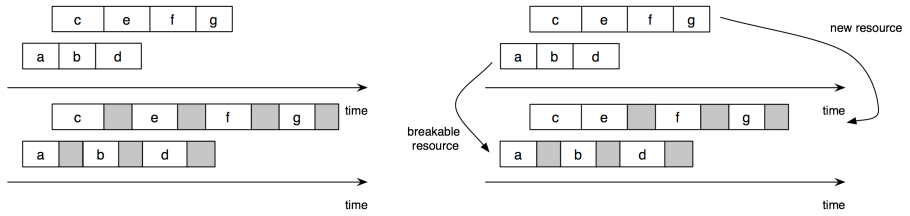


Fig. 2 Adding slack time under the Time Windows Slack and the Focused Time Window Slack methods.

The Focused Time Window Slack (FTWS) approach was also proposed in [4]. In this approach, the placement of activities in the scheduling horizon is taken into account in order to decide if some slack time will be needed. The authors use uncertainty statistics in order to focus the slack in parts of the schedule that are more likely to need it in order to deal with an unexpected event. For example, they consider the scenario where a new machine can arrive to the system, in this case it will not make sense to make the schedule more robust at the beginning, as this machine is new and it could take some time before it can break down. The slack for the activities is calculated as a function of the probability that a breakdown will occur before or during its execution. Which means that the decision on whether to add slack or not is based on known distributions. Figure 2 (right-hand side) shows how the FTWS method adds the extra execution time. It is possible to see that slack is not added in the first operation scheduled on the new resource.

In general, all these slack-based methods base their behavior on the data they receive about the uncertainty, but what if something happens on a machine where breakdowns are not expected? Consider again the example of the newly bought machine, a very fast one which is not considered as breakable, therefore, several operations are scheduled there, because it is a more reliable resource. If that machine fails, the schedule is not prepared to deal with the disruption, as it was a breakdown impossible to forecast. This leads us to the following question:

Is it possible to construct more robust schedules without looking at the probability distributions of the unexpected events as the only source of information? In the next section we propose a new slack-based approach which does not rely on the probability distributions as the only source of information. It works by analyzing something we call 'criticality' of the machines and the main objective is to add slack times in those parts of the schedule which are considered to be critical, or more likely to need extra execution time.

5 Robust approach based on 'criticality'

In this section we propose a new way to incorporate slack into the schedule, the objective is to gain in robustness without loosing too much in terms of optimality. In order to do this we need to define an important concept, which is the 'criticality' of the machines. The well-known job shop scheduling instance ft06 will be used as example, for which Figure 3 shows the optimal solution (Gantt chart) in terms of makespan, which is $C_{max} = 55$. In this figure, every job is represented by a color, each block is an operation and the number inside the block is the id of the operation within the corresponding job.

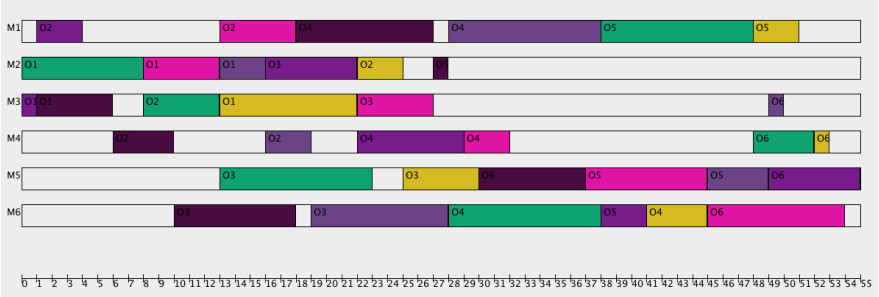


Fig. 3 Optimal solution for the instance ft06, $C_{max} = 55$

Analyzing this optimal solution, we see that there are two machines that remain busy during the initial part of the scheduling process, these are M_2 and M_3, while there are other machines, like M_5 and M_6 with a higher workload at the end. The occurrence of an unexpected event, for example, a temporary breakdown on machine M_1 or on M_2 at the beginning of the scheduling period will not have the same impact as a breakdown on M_4. This is due to the fact that M_4 has some idle time in between the different operations, which will allow to absorb part of the breakdown. In other words, it is not a critical machine. This analysis leads us to the following definition:

Definition 1 (Critical Machine) A machine M_i is said to be critical between time t_x and time t_y, if the number of consecutive operations scheduled on that interval is higher than a threshold Z.

Of course this threshold is a sensitive parameter that will depend on the problem being solved, and which needs to be carefully selected. The user can define when a machine is considered as critical through this threshold and as such, decide on the level of robustness of the schedules generated by the algorithm. In case it is not defined by the user, the approach calculates this threshold depending on the total number of operations that have to be executed by the machine.

Let us assume that the system for which the optimal schedule was proposed in figure 3 will be under a series of perturbations. Under normal circumstances, without having any previous knowledge about the problem at hand, not having any proactive or reactive technique, parallel machines or other solution method, the only choice is to wait for the perturbations to finish in order to resume the execution of the operations

on the affected machines. If we do that for the optimal solution presented in Figure 3, then we will obtain the schedule shown in Figure 4, for which the makespan is $C_{max} = 64$. In this case we performed a simple 'right shift' every time a perturbation occurred. Perturbations are represented by a block with yellow and red horizontal lines.

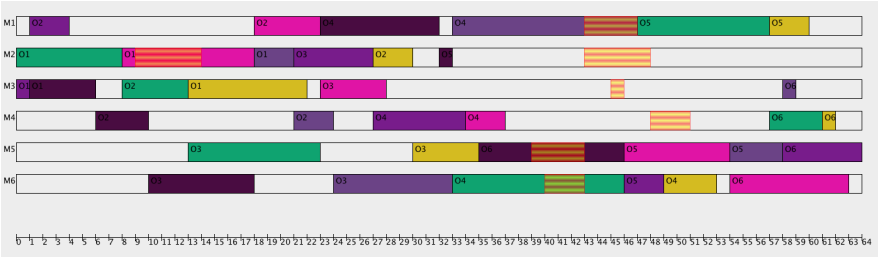


Fig. 4 Optimal schedule affected by several perturbations, $C_{max} = 64$.

Let us say that the makespan refers to the number of days that it will take to finish the execution of the whole schedule, which is the completion time of an order from an important client. If we compare the two previously shown solutions, we will see that the user will have to wait for 9 extra days in order to get his order.

If we use one of the approaches described in the previous section, for example, temporal protection, we could construct a more robust schedule by adding slack times to those operations that are scheduled on breakable resources, which in this case, according to the perturbations shown in Figure 4, will be in all the operations, as all the machines will be affected. This will result in a very robust schedule, but with a very low quality. It is important to remember that in these existing slack-based approaches it is assumed that the information concerning the breakdowns, such as the mean down times of the machines, etc, is known. In this case the processing of the order will be expected in 90 days, $C_{max} = 90$.

It is clear that it is important to properly manage the slack times in order to obtain a solution that is robust enough to deal with unexpected events, but good in terms of quality at the same time. There are two important elements to define: in which part of the schedule to add the slack times, and how long should they be. Here is where the criticality defined in the previous section plays an important role. The approach we propose is based on this definition (Definition 1), as we do not rely on the probability distributions and the information about the machines, but on their workload. If a machine is considered as critical, then the operations scheduled during the critical period will receive an extra amount of processing time.

The process goes as follows: We execute our RL algorithm in order to find a near-optimal solution, the slack-based algorithm analyzes which are the critical machines, and then adds slack times to those operations scheduled on the critical intervals of the resources. As for the amount of extra time to add, we consider that it is something problem dependent, therefore, we give the user the possibility to define it. However, the approach can also compute a value for it based on the known information about the machines. Our objective is to change the way of defining the places where the slack is added, focusing more on critical parts of the solution. In order to compare with the

existing techniques, the amount of slack to add is the same for all the methods, defined as a percentage of the processing time of the corresponding operation.

Figure 5 shows an example of a schedule constructed using the proposed approach, which means that slack times are only added to the operations scheduled on critical resources. In this case a machine is considered as critical if the number of consecutive operations is higher than three (machines M_2, M_5 and M_6 in this example, see Figure 3 to compare). The makespan obtained after this process is $C_{max} = 69$. Slack times are represented by a horizontal black line and are added after each block.

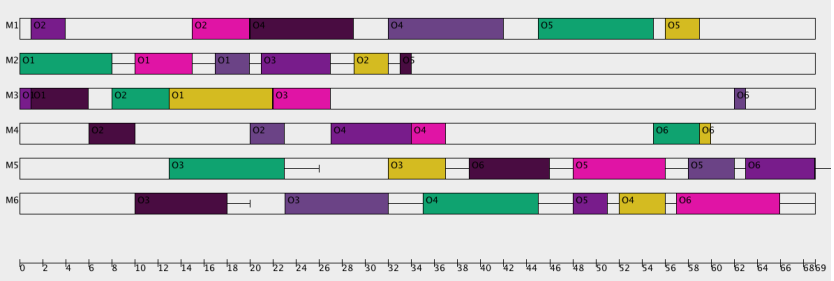


Fig. 5 Schedule obtained by adding slack times on the critical machines after the scheduling process.

An advantage of the proposed approach is that it allows to define a starting point for the slack times to be added. For example, if the user considers that slack will only be needed after half of the process has passed, this is a constraint that can be easily satisfied. We also implemented a second way to add slack to the operations scheduled on critical resources. This second way is also based on the criticality of the machines, but instead of taking an already constructed schedule and analyze which are the critical resources, it does the same analysis during the scheduling process. This means that if a machine is identified as critical, the processing times of the operations scheduled on the machine will be extended, and once this new processing time is calculated, it is used to allocate the operations on the machines. This second alternative (i.e. adding slack during the scheduling process) gives the possibility of allocating the operations in a better way once the critical resources have been identified.

Figure 6 shows an example of a schedule constructed using this second alternative. If we compare it with the schedule obtained by the previous alternative (Figure 5, $C_{max} = 69$), the one depicted in this figure shows how a better schedule (in terms of makespan) can be obtained if the last two operations scheduled on M_2 are swapped. In this case the makespan is $C_{max} = 66$.

6 Experimental Results

In this section we will take into account the deviation from the original schedule in order to compare different schedules in terms of robustness. Figure 7 shows a comparison between a non-protected schedule and two schedules generated using our approach based on criticality. The left hand side shows the comparison with a schedule where

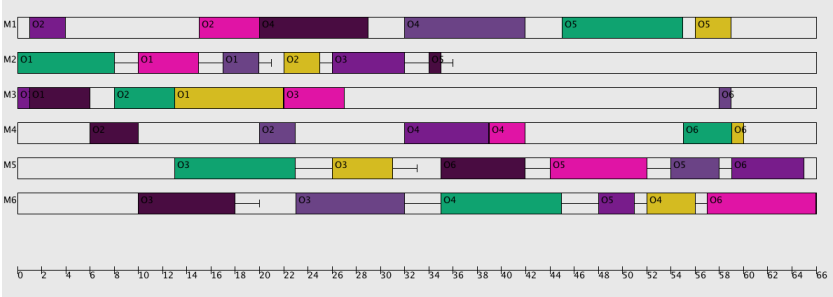


Fig. 6 Schedule obtained by adding slack times on the critical machines during the scheduling process

the slack was added during the scheduling process and the one in the right hand side was obtained by adding slack after generating the solution. A non-protected schedule is a standard solution obtained by any method which does not take uncertainty into account, it could even be an optimal solution. In this case we use a schedule that is constructed by our basic algorithm, described in section 2. These schedules are compared in terms of the deviation of its execution from the originally proposed schedule.

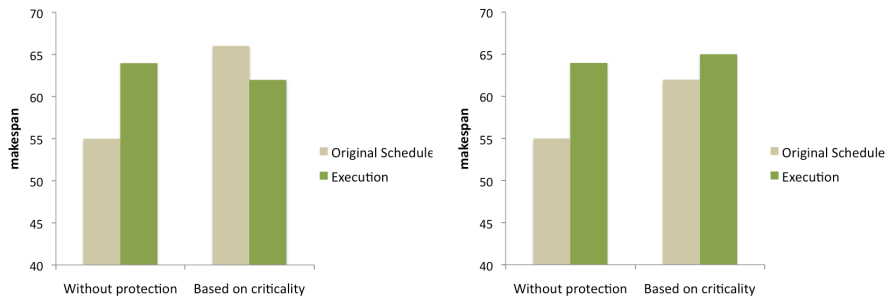


Fig. 7 Comparing the execution of non-protected schedules and solutions constructed using our approach based on criticality, in terms of the deviation from the original schedules.

From Figure 7 we can see that the non-protected schedule was expecting a performance of $C_{max} = 55$, while the real execution was $C_{max} = 64$. The approach based on criticality is able to propose schedules which are closer in time to their real executions. The left hand side of the figure shows a schedule with a deviation of 4 time units and the right hand side a schedule with a deviation of 3 time units, while the deviation of the non-protected schedule was 9. We can notice that the alternative of adding slack while constructing the schedule can, in some occasions, obtain schedules with a better execution time compared to the non-protected solutions.

Let us assume that we want to solve a scheduling problem, for which we know the number of machines and the number of jobs (with their corresponding operations and processing times). The mean down time and mean time between failures of the machines are also provided. In this setting all the machines are considered as breakable, meaning

that they are expected to have at least one perturbation. This is how the different algorithms will work:

- Without protection: Solves the problem looking for the optimal solution, in this case the objective is to minimize the makespan.
- Temporal protection: Checks for the breakable machines and adds slack times to all the operations scheduled on those resources.
- Based on criticality: Solves the problem and analyzes which are the critical machines in the solution found, then adds slack times in three possible ways:
  1. to all the operations scheduled on critical resources (option executed by default);
  2. to those operations scheduled on the critical period(s) of the resource;
  3. to those operations that are scheduled after a specific moment in time.

This could lead us to an outcome like the one shown in Figure 8. The temporal protection method is expecting several breakdowns, that is why its proposed schedule has a very high makespan. If we search for a solution without protection, then disruptions are not expected, therefore, the proposed schedule is the lowest in makespan. If we go for the criticality-based approach, we find a combination of these two behaviors. We add extra time but we do it based on the workload of the machines, instead of using the breakdown probabilities like the temporal protection method, which results in a better balance between optimality and robustness.

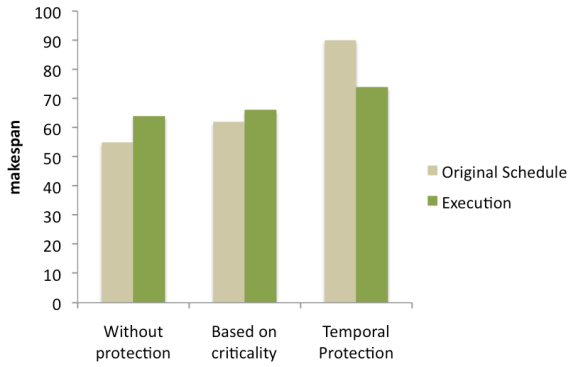


Fig. 8 Measuring the deviation from the original schedules.

Unfortunately, there are no genuine benchmark problems as the aspect of robustness is not well defined. The approach taken in this paper is similar to the one taken in most papers, where the instances are problems for which the robust schedule obtained by the algorithm is evaluated under 'disrupted' conditions. In order to evaluate our approach we consider instances that are executed with the resulting schedule assuming different levels of perturbations (1 to 6). By level of perturbations we mean the number of unexpected events that can occur during the scheduling period. This amount increases with the level, meaning that a level 6 instance has more perturbations than an instance level 1. We used the classical JSSP instances from the OR-Library and added perturbations using the Erlang distribution for the mean inter-arrival time and Poisson for the durations. Figure 9 shows a comparison between the approaches according to their deviation from their original schedule using these 6 levels of perturbations.

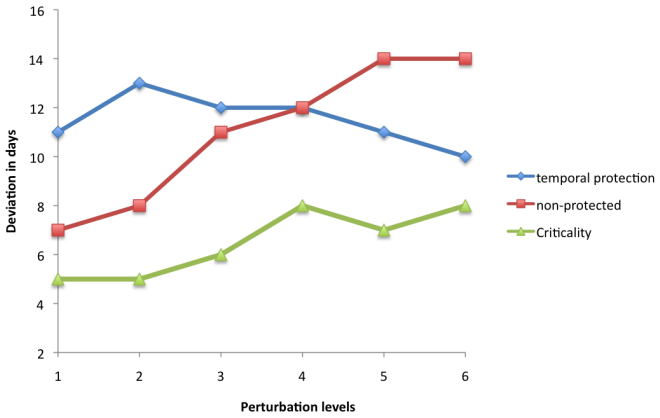


Fig. 9 Comparison between the approaches according to their deviation from their original schedules using six levels of perturbations.

The higher the uncertainty, the worse the performance of the non-protected schedule. This is of course due to the fact that it was not prepared to deal with unexpected events. In the case of the temporal protection technique, it starts with a high deviation from the original schedule, but it gets better results as the level of perturbations increases, this is because the amount of slack time included in the solution when using this technique is usually high. Our approach (criticality) showed a more 'stable' behavior, it was able to keep the same result for the first two levels of uncertainty, but once the unexpected events started to increase it was also affected, although it was still able to keep the lowest deviation from the original schedule.

7 Conclusions

Scheduling is an active research field in which several important challenges arise, for example, dealing with uncertainty is a topic that is recently receiving more attention. It is common to face unexpected events in scheduling environments, specially in manufacturing, where machines can break down, orders can take longer than expected and all these events can make the original schedule fail. That is why it is preferable to have a solution that is robust enough to absorb some level of uncertainty, instead of a possibly optimal one, which will fail in the presence of any small disruption. In this paper we presented a new approach that can incorporate some robustness in the solutions. The proposed approach allows the user to set, in a transparent way, certain parameters involved in the solution construction process, in order to define the balance between robustness and optimality.

In general, we receive a scheduling problem which depending on its characteristics can be solved using one of the existing RL algorithms and will incorporate the proposed slack-based approach. This will provide as output a near-optimal schedule, which can have different levels of robustness. It is also possible to identify which are the critical machines and provide a visualization of the resulting schedule in the form of a Gantt chart. This visualization helps the user to carefully analyze the proposed solution, and

maybe change some parameter settings in order to analyze what will happen if the system is under different circumstances.

References

1. Beck J, Wilson N (2007) Proactive algorithms for job shop scheduling with probabilistic durations. Journal of Artificial Intelligence Research 28(1):183–232
2. Chiang W, Fox M (1990) Protection against uncertainty in a deterministic schedule. In: Proceedings 4th International Conference on Expert Systems and the Leading Edge in Production and Operations Management, pp 184–197
3. Davenport A, Beck J (2000) A survey of techniques for scheduling with uncertainty. Unpublished manuscript p 49
4. Davenport A, Gefflot C, Beck JC (2001) Slack-based techniques for robust schedules. In: Proceedings of the Sixth European Conference on Planning (ECP-2001)
5. Gabel T (2009) Multi-agent reinforcement learning approaches for distributed job-shop scheduling problems. PhD thesis, Universität Osnabrück
6. Gabel T, Riedmiller M (2007) On a successful application of multi-agent reinforcement learning to operations research benchmarks. IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning pp 68–75
7. Gabel T, Riedmiller M (2007) On a Successful Application of Multi-Agent Reinforcement Learning to Operations Research Benchmarks. In: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, Honolulu , USA, pp 68–75
8. Gao H (1996) Building robust schedules using temporal protection-an empirical study of constraint based scheduling under machine failure uncertainty. Master thesis, University of Toronto
9. Herroelen W, Leus R (2005) Project scheduling under uncertainty: Survey and research potentials. European Journal of Operational Research 165(2):289–306
10. Le Pape C (1991) Constraint propagation in planning and scheduling. Tech. rep., Standford University
11. Martínez Y (2012) A generic multi-agent reinforcement learning approach for scheduling problems. PhD thesis, Vrije Universiteit Brussel
12. Martínez Y, Vreckem B, Catteeuw D, Nowé A (2010) Application of learning automata for stochastic online scheduling. In: Diehl M, Glineur F, Jarlebring E, Michiels W (eds) Recent Advances in Optimization and its Applications in Engineering, Springer, pp 491–498
13. Martínez Y, Nowé A, Suárez J, Bello R (2011) A reinforcement learning approach for the flexible job shop scheduling problem. Lecture Notes in Computer Science Proceedings of the fifth International Conference on Learning and Intelligent Optimization(LION5) Vol. 6683:pp. 253–262
14. Pinedo M (2008) Scheduling: theory, algorithms, and systems, Third Edition. Springer Verlag
15. Policella N (2005) Scheduling with uncertainty a proactive approach using partial order schedules. Phd thesis, Università degli Studi di Roma "La Sapienza"
16. Sutton RS, Barto AG (1998) Reinforcement Learning: An Introduction. The MIT Press
17. Zhang W (1996) Reinforcement learning for job shop scheduling. PhD thesis, Oregon State University

MISTA 2013

Extended Roster Quality Staffing Problem

A medium term roster quality to support staffing decisions

Komarudin · Tim De Feyter · Marie-Anne Guerry · Greet Vanden Berghe

Abstract This paper is concerned with integrating the roster quality and the staffing decisions. In particular, the roster quality is aggregated over time to match the staffing planning period. It is in contrast with previous research that relied on the quality of a roster representing a much shorter horizon than the staffing period. This model extends on previous work by providing more accurate staffing decisions based on roster quality for the whole staffing period and taking into account all rostering constraints. The roster quality is obtained by a scenario-based aggregation. A simulated annealing has been applied to the model. The method has been tested on several problem instances, and the results show that it can produce a better personnel structure in terms of the roster quality than the initial personnel structure.

Keywords: personnel structure optimization, staffing, rostering, expensive optimization.

1 Introduction

Staffing aims at determining the most suitable personnel structure, which is adequate to cover the workload, and avoids to outnumber the required number of employees. The personnel structure represents the number of employees in distinct personnel subgroups, distinguished by specific work-related and individual characteristics of its members (e.g. skills, contractual agreements). The personnel in a subgroup are considered homogeneous according to staffing criteria. For accurate staffing, besides workload, also other factors should be considered. Personnel absenteeism (due to illness, days off or vacation) and the need for personnel rotation between shifts and workplaces, for

Komarudin · Marie-Anne Guerry
Vrije Universiteit Brussel, Centre for Manpower Planning and Studies
E-mail: {komarudin, maguerry}@vub.ac.be

Tim De Feyter
Hogeschool-Universiteit Brussel, Center for Business Management Research
E-mail: tim.defeyter@hubrussel.be

Greet Vanden Berghe
KAHO Sint-Lieven, Combinatorial Optimisation and Decision Support
KU Leuven, Department of Computer Science
E-mail: greet.vandenberghe@kahosl.be

example, increase the required number of employees for handling the workload. Such restrictions are formally described as the constraints in the personnel rostering problem (Bilgin et al., 2012).

Several previous research contributions consider staffing problems while taking rostering requirements into account. While Haspeslagh (2012) proposes a negotiation protocol to allocate the available nurses to multiple wards, Maenhout and Vanhoucke (2013) suggest a branch and price method. Both approaches include coverage requirements, contractual as well as personal constraints. Although their methods ensure higher roster quality and more accurate staffing decisions than methods that neglect the rostering constraints, they determine the staffing decision based on short term roster quality. However, in general, staffing and rostering span a different planning period. One staffing period usually consists of several rostering periods. Therefore, in evaluating staffing decisions, the quality of the personnel rosters over the whole staffing period must be taken into account.

In order to consider short term roster quality in medium term staffing, some academic contributions integrate staffing with cyclic rostering. Isken and Hancock (1998) introduce a one-week cyclic rostering model that intends to minimize the understaffing cost. Based on the model, they examine the suitability of several staffing scenarios. Bard and Purnomo (2006), on the other hand, present a nurse addition problem which aims to determine the number of additional workers needed in satisfying a cyclic rostering model. Likewise, Beliën et al. (2012) propose an approach to solve an integrated staffing and cyclic rostering problem for airline maintenance. These models are suitable for problems that have high stability over the staffing period such that cyclic rosters can be used repeatedly. However, they may be unsuitable for problems that exhibit demand and personnel behavior variability among the rostering periods within the same staffing period (De Causmaecker and Vanden Berghe, 2003). Demand variability largely originates from customer behaviors. For example, on a yearly staffing period, health-care service demands are usually higher in winter than in summer. Personnel behavior, on the other hand, can vary from one rostering period to another, for example, because of more days-off requests during school holiday periods.

Considering only one rostering period in supporting staffing decisions, as in Haspeslagh (2012) and Maenhout and Vanhoucke (2013), can be problematic for staffing accuracy because it neglects the demand and personnel behavior variabilities among rostering periods. Several strategies are possible to tackle this difference in time horizon between staffing and rostering. One can select the rostering period with the highest or the lowest workload in determining the personnel structure. Although using the rostering period with the highest workload can guarantee that the personnel is adequate for the other rostering periods, it can result in a very expensive personnel cost. Utilizing the rostering period with the lowest workload may not be an efficient option either, since it can increase overtime and/or temporary personnel costs. Another strategy could be to change the personnel structure from one rostering period to the next period. This may not be desirable since external personnel may not always be available for immediate recruitment. Moreover, it increases recruitment and lay-off costs, and could harm personnel motivation (Larrabee et al., 2003).

A more appropriate alternative is the flexible staffing strategy (Ozcan, 2009). In this case, the personnel structure is established based on long term rostering considerations, accompanied by short term adjustments when necessary. Several scholars already applied the flexible staffing strategy. Venkataraman and Brusco (1996) introduce an iterated approach of two integer programs, each for staffing and rostering. The person-

nel structure is determined in the staffing phase, and then the personnel is scheduled based on several prescribed working patterns. The rostering phase only considers a restricted set of the feasible working patterns.

More recently, Mundschenk and Drexl (2007) and Li et al. (2007) introduced a staffing problem while considering rostering constraints. Mundschenk and Drexl (2007) aim at meeting the aggregated coverage requirements throughout the year. Li et al. (2007) constrain the maximum amount of overtime and the maximum number of days-off. The problem considered in the present paper involves more general rostering constraints. We propose an approach for staffing while explicitly considering the predicted roster quality over the whole staffing period. Particularly, our method aims to provide more accurate staffing decisions by producing rosters for each member of the personnel, as suggested by Chan et al. (2006). Therefore, we will use the formulation of the roster quality staffing (RQS) problem, which aims to obtain a personnel structure that maximizes the roster quality (Komarudin et al., 2013), considering not only the coverage requirements, but also contractual and personal constraints. In Section 2, by applying the flexible staffing strategy, we propose an extension of the original RQS problem that did not yet consider the difference in time horizon between staffing and rostering. Particularly, the present paper aggregates the roster quality over the whole staffing period. As is shown in Section 2, the extended RQS problem can be categorized as a stochastic black-box expensive optimization problem (Huang et al., 2006). The response surface methodology (RSM) is one of the common methods to solve such expensive optimization problems (Talbi, 2009). However, RSM may not be suitable for the extended RQS problem, because large number of samples need to be evaluated while the evaluation of the roster quality requires substantial amount of computation time. Therefore, in Section 3, besides the RSM method, we will propose a simulated annealing (SA) approach for the extended RQS problem. In Section 4, data set are provided to test the proposed algorithms. In Section 5, we discuss the analyses of the considered algorithms and compare the results of the SA and RSM methodology. Finally, Section 6 provides the conclusion and future research directions.

2 The Extended Roster Quality Staffing problem

This section first describes the Roster Quality Staffing problem formulated in Komarudin et al. (2013). The problem does not consider the difference in planning period of staffing and rostering. Afterwards, the extended RQS problem which aggregates the roster quality over the staffing period is explained.

A categorization is considered to divide the personnel into several subgroups that are homogeneous with respect to individual and/or work related characteristics. For example, a subgroup could consist of all the regular nurses with full time contracts and Wednesday afternoons off. The number of members of subgroup i is denoted as n_i. Let k be the number of subgroups, then the personnel structure n is defined as in Eq. 1.

$$n = [\, n_1 \; n_2 \; ... \; n_k \,] \tag{1}$$

Assume an algorithm P that generates a solution to a rostering problem. A roster generated by algorithm P, given personnel structure n, is denoted by $R(n)$. The quality of roster $R(n)$ is characterized by the degree to which the soft constraints of the roster problem are violated. The quality violation $qv(R(n))$ of roster $R(n)$ represents the

weighted sum of violations of the soft constraints (Bilgin et al., 2012). As in Komarudin et al. (2013), the objective of the current paper is to obtain $n^* \in Q$ that optimizes the roster quality over the staffing period, with Q being the set of personnel structures that satisfy the staffing constraints.

In searching for n^*, we need to explore alternative personnel structures $n' \in Q$, by comparing their roster quality. However, computing the roster quality for n' is not as straightforward as for the current personnel structure. A subgroup is homogeneous with respect to staffing characteristics but the members may differ individually in terms of rostering characteristics. For the current personnel structure, those constraints are known. This allows generating a roster and computing the corresponding quality violations. In the staffing phase, on the other hand, the future members of the organization (and their individual rostering characteristics) are not yet known. Therefore, to assess $qv(R(n'))$, it is first necessary to obtain problem instances that fairly represent the characteristics of future employees.

There are several possibilities to obtain a set of members in subgroup i corresponding with the number n'_i. A personnel structure n' is a modification of n. From the members corresponding with the initial n_i, we can get the members corresponding with n'_i by randomly duplicating or removing some members of subgroup i.

The value of $qv(R(n'))$ may not be a good representation of the roster quality of the personnel structure n'. The rostering problems that correspond with the personnel structure n' are not unique since duplicating and removing members in a certain subgroup is performed randomly. Consequently, the rostering problems may contain the rostering constraints that are not in common although they correspond with a same n'. For example, a rostering problem involves a duplicate of worker X who prefers free Wednesday afternoons while another rostering problem involves a duplicate of worker Y who prefers free Friday night, although both rostering problems correspond to the same n'. Moreover, algorithm P can also have non-deterministic elements to solve the rostering problem. That is, if the use of algorithm P is repeated, it can result in different rosters. As a result, we suggest to use the average of the roster quality $\bar{qv}(R(n'))$ to represent the roster quality of n', instead of $qv(R(n'))$, as in Komarudin et al. (2013). The value of $\bar{qv}(R(n'))$ can be obtained by producing m^{opt} rostering problem instances corresponding with n' and solving them using algorithm P. In this paper, we define $\bar{qv}(R(n'))$ as the arithmetic mean of $\{qv(R(n'))_1, qv(R(n'))_2, ..., qv(R(n'))_{m^{opt}}\}$.

We formulate the extended RQS in order to deal with the difference in planning period of staffing and rostering. The next staffing planning period is considered to be $[T_0^s, T_1^s]$. The notation $[T_{t-1}^r, T_t^r]$ denotes the rostering planning period t. Let one staffing period correspond with m_r rostering periods. For example, if a staffing period lasts for one year with $m_r = 12$, then the rostering is performed each month.

We are interested in finding the most suitable staffing decision $n' \in Q$ for the next staffing period (i.e., from T_0^s to T_1^s). Considering a flexible staffing strategy, the personnel structure is assumed to be fixed during the staffing planning period. In order to support a staffing decision from T_0^s to T_1^s, the roster quality for the same period, i.e. from T_0^r to $T_{m_r}^r$ needs to be calculated. We denote $\bar{qv}_{T_0^s, T_1^s}(R(n'))$ as the roster quality of personnel structure n' for the planning period $[T_0^s, T_1^s]$. We calculate $\bar{qv}_{T_0^s, T_1^s}(R(n'))$ as in Eq. 2. Other alternatives can be used to calculate $\bar{qv}_{T_0^s, T_1^s}(R(n'))$, such as the maximum $\bar{qv}_{T_t^r, T_{t+1}^r}(R(n'))$.

$$\bar{qv}_{T_0^s,T_1^s}(R(n')) = \frac{1}{m_r} \sum_{t=0}^{m_r-1} \bar{qv}_{T_t^r,T_{t+1}^r}(R(n')) \quad (2)$$

If the rostering problems according to the planning period $[T_0^r, T_{m_r}^r]$ can be grouped based on their rostering constraints, then $\bar{qv}_{T_0^s,T_1^s}(R(n'))$ should be calculated based on these groups. The groups with similar constraint sets are called scenarios (e.g. a winter scenario with high workload and high personnel absenteism, or a summer scenario with low workload and low absenteism). Scenario-based aggregation to calculate $\bar{qv}_{T_0^s,T_1^s}(R(n'))$ is more efficient if the number of scenarios m^{sc}, is less than m_r. In order to calculate $\bar{qv}_{T_0^s,T_1^s}(R(n'))$, we only need to solve m^{sc} rostering problems instead of m_r.

Let ϕ_i be the probability of scenario i for the planning period $[T_0^r, T_{m_r}^r]$ and $\bar{qv}(R(n'))_i$ the roster quality of scenario i with respect to one rostering period, then $\bar{qv}_{T_0^s,T_1^s}(R(n'))$ in Eq. 3 can be regarded as the roster quality of personnel structure n' that takes into account a medium planning period and several scenarios (such as absenteeism and work-overload situation). ϕ_i should represent the relative frequency of scenario i compared to other scenarios satisfying $\sum_{i=1}^{m^{sc}} \phi_i = 1$. Moreover, ϕ_i can be replaced by a more general weight which represents additional considerations such as the relative importance of scenario i to other scenarios.

$$\bar{qv}_{T_0^s,T_1^s}(R(n')) = \sum_{i=1}^{m^{sc}} \phi_i \bar{qv}(R(n'))_i \quad (3)$$

RQS aims at finding the most preferable personnel structure $n^* \in Q$ such that:

$$\bar{qv}_{T_0^s,T_1^s}(R(n^*)) \leq \bar{qv}_{T_0^s,T_1^s}(R(n')) \; \forall n' \in Q \quad (4)$$

The RQS problem can be categorized as a stochastic black-box expensive optimization problem. It is stochastic because the measured output contains random error (Huang et al., 2006). Particularly, a personnel structure can produce multiple rostering problems and it eventually produces different values of roster quality. The evaluation of the roster quality is a black box function which cannot be substituted by a simple analytical function (Talbi, 2009). The RQS problem is an expensive optimization problem because the evaluation of roster quality requires substantial amount of computation time.

We assume the number of members in the personnel structure are restricted by prespecified lower and upper bounds. We denote n_i^{LB} and n_i^{UB} as the lower and the upper bound of the number of members of subgroup i respectively, $n_i^{LB} \leq n_i' \leq n_i^{UB}$.

3 Algorithms for Solving Roster Quality Staffing Problems

One common method to deal with an expensive optimization problem is RSM. We explain the RSM to solve the roster quality staffing problem in this section. Moreover, we propose SA which has a lower computation time per-iteration than RSM to enable to solve large roster quality staffing problems. The method to generate a rostering problem that corresponds with a certain personnel structure is taken from Komarudin et al. (2013).

SA and RSM evaluate the neighboring personnel structure n' in order to obtain n^*. For calculating $\bar{qv}_{T_0^s,T_1^s}(R(n'))$, we need to solve $m^{opt}m^{sc}$ rostering problem instances. This can be expensive and can also prevent the algorithm from converging. In order to avoid to explore unpromising neighboring personnel structures and to deal with the expensive computation of $\bar{qv}_{T_0^s,T_1^s}(R(n'))$, we use two modifications both in RSM and SA:

- Delay the calculation of $\bar{qv}_{T_0^s,T_1^s}(R(n'))$.
  We do not always calculate $\bar{qv}_{T_0^s,T_1^s}(R(n'))$. Instead, we only calculate $qv_{T_0^s,T_1^s}(R(n'))$ when evaluating n'. Specifically, we only solve m^{sc} rostering problem instances to evaluate n'. However, if n' is promising, we fully evaluate $\bar{qv}_{T_0^s,T_1^s}(R(n'))$ to check if it can be better than the best-so-far solution. In this way, we can save computation time to be used for subsequent iterations.
- Save several promising personnel structures
  If we only rely on the first modification, there is a chance that we neglect a promising personnel structure just because the value of $qv_{T_0^s,T_1^s}(R(n'))$ is a little higher than $\bar{qv}_{T_0^s,T_1^s}(R(n^*))$. However, if we fully evaluate n', $\bar{qv}_{T_0^s,T_1^s}(R(n'))$ may be better than $\bar{qv}_{T_0^s,T_1^s}(R(n^*))$. Therefore, we save a number of promising personnel structures in the set $\mathbb{P}^{save}$ and we restrict the number to be m^{save}. Whenever n' is identified as a promising personnel structure, we update $\mathbb{P}^{save}$. After the iteration of RSM or SA has ended, $\bar{qv}_{T_0^s,T_1^s}(R(n))$ for each $n \in \mathbb{P}^{save}$ is calculated. We update n^* if the $n \in \mathbb{P}^{save}$ is proven to be better than the best-so-far solution.

3.1 The Response Surface Methodology

The response surface methodology (RSM) utilizes a sequence of designed experiments to achieve an optimal response. In our case, RSM is intended to find the personnel structure n that minimizes $\bar{qv}_{T_0^s,T_1^s}(R(n))$. For each designed experiments, RSM picks a number of samples $\mathbb{P}^{samples}$ in the neighborhood of the current personnel structure and generates a surrogate function $F(n)$. $F(n)$ is used to mimic the black-box function $qv_{T_0^s,T_1^s}(R(n))$ and it is supposed to yield the best neighboring personnel structure $n' \in \mathbb{P}^{samples}$. $F(n)$ is used rather than $qv_{T_0^s,T_1^s}(R(n))$ because it is usually a simple function and can be evaluated very quickly. RSM is iterated until a stopping condition is met. RSM can use several stopping conditions including the maximum computation time, the maximum number of iterations, etc. For further discussion about RSM, see e.g. Montgomery (2001).

In the current paper, $F(n)$ is estimated by considering linear and quadratic relations between n_i as the explanatory variable and $qv_{T_0^s,T_1^s}(R(n))$ as the response variable. Specifically, the explanatory variables consist of n_i, n_i^2 and $n_i n_j$. The linear relationship between n_i and the roster quality is considered because some constraint violations (e.g. series constraint violations) can grow linearly as n_i increases or decreases. We include a quadratic relationship because some constraint violations can have a curved relationship with n_i (Komarudin et al., 2013). Examples of this type of constraint are the coverage requirement and the contractual hours constraint. Moreover, we also include $n_i n_j$ interaction because one subgroup can have a relationship with another in form of overlapping rostering constraints (Komarudin et al., 2013). Examples of this type of constraint are coverage requirements with overlapping skills, tutorship etc.

In our model, $F(n)$ is represented in Eq. 5. θ_0 is the intercept coefficient of the surrogate function F(n) and θ_i^l represents the linear coefficient of the surrogate function for subgroup i. θ_i^q represents the quadratic coefficient of the surrogate function for subgroup i and θ_{ij}^i represents the interaction coefficient of the surrogate function for subgroup i and subgroup j. The coefficients of $F(n)$ are obtained by applying a linear regression method. Specifically, we employ STEP and RLM (robust linear model) to enhance the model accuracy (Venables and Ripley, 2002). STEP is used to modify the base model in Eq. 5 to only include explanatory variables that have a significant relation with the roster quality. RLM uses iterated least squares to decrease the effect of influential data points (such as outliers).

$$F(n) = \theta_0 + \sum_{i=1}^{k} \theta_i^l n_i + \sum_{i=1}^{k} \theta_i^q n_i^2 + \sum_{i=1}^{k} \sum_{j=i+1}^{k} \theta_{ij}^i n_i n_j \tag{5}$$

RSM algorithm is shown in Algorithm 1. It starts with three personnel structure sets, i.e. $\mathbb{P}^{save}$, $\mathbb{P}^{hist}$ and $\mathbb{P}^{samples}$. $\mathbb{P}^{save}$ is used to keep promising personnel structures to be evaluated after the iteration has ended. $\mathbb{P}^{hist}$ contains all the personnel structures which have been evaluated, i.e. the $qv_{T_0^s,T_1^s}(R(n))$ values. RLM uses the data in $\mathbb{P}^{hist}$ to help establishing the surrogate function $F(n)$. $\mathbb{P}^{samples}$ consists of the personnel structures that are generated in each iteration.

Iteratively, Line 4 generates $\mathbb{P}^{samples}$ using optimal design (OD) (Montgomery, 2001). The personnel structures in $\mathbb{P}^{samples}$ are restricted to the neighborhood of the best-so-far personnel structure n^* by specifying the value of m^{range}. m^{range} is a positive integer number to control the size of the neighborhood of the personnel structure. Afterwards, $F(n)$ is estimated using RLM and STEP utilizing personnel structure data $\mathbb{P}^{samples}$ and $\mathbb{P}^{hist}$ that are in the neighborhood of n^* in Line 5. The particle swarm optimization (PSO) approach makes use of the surrogate function $F(n)$ to obtain a good neighboring personnel structure n' in Line 6. However, we also check whether a personnel structure in $\mathbb{P}^{samples}$ can replace the one from PSO to be n' in Line 8. Lines 9-14 evaluate whether n' can replace n^*. Line 10 computes the actual roster quality. Lines 15-18 save only m^{save} personnel structures to be evaluated after the iteration finish. Finally, Lines 20-21 select from n^* and $\mathbb{P}^{save}$ the personnel structure that has the best roster quality.

In addition to two modifications explained in the beginning of Section 3, we use several modifications to RSM:

- Increase the neighborhood area if the algorithm converges prematurely
  $\mathbb{P}^{samples}$ can have a larger neighborhood by increasing the value of m^{range}. This modification is used to help RSM when it cannot find a new better n^* after a few number of iterations. The value m^{range} is restored to the original if RSM can find a new better n^*.
- Personnel structures from the previous iterations are included to estimate $F(n)$
  $\mathbb{P}^{hist}$ keeps data from earlier iterations to help estimating $F(n)$. However, the algorithm only includes the personnel structures in the neighborhood of n^*.

Input: initial solution n^0;
$\mathbb{P}^{save} := \{\}$; $\mathbb{P}^{hist} := \{\}$; $\mathbb{P}^{samples} := \{\}$;
while *stopping condition not met* **do**
  Generate $\mathbb{P}^{samples}$ using optimal design;
  $\mathbb{P}^{samples} := \{n | max(n_i^{LB}, n_i^* - m^{range}) \leq n_i \leq min(n_i^{UB}, n_i^* + m^{range}\}$;
  Establish surrogate function $F(n)$ using
  $\{n | n \in \mathbb{P}^{hist} \cup \mathbb{P}^{samples}; max(n_i^{LB}, n_i^* - m^{range}) \leq n_i \leq min(n_i^{UB}, n_i^* + m^{range}\}$;
  Obtain the best-move n' using PSO that minimizes $F(n')$;
  Calculate $qv_{T_0^s,T_1^s}(R(n'))$;
  $n' := \underset{n \in (\mathbb{P}^{samples} \cup n')}{arg\ min} (qv_{T_0^s,T_1^s}(R(n)))$;
  if $qv_{T_0^s,T_1^s}(R(n')) < \bar{qv}_{T_0^s,T_1^s}(R(n^*))$ **then**
    Calculate $\bar{qv}_{T_0^s,T_1^s}(R(n'))$;
    if $\bar{qv}_{T_0^s,T_1^s}(R(n')) < \bar{qv}_{T_0^s,T_1^s}(R(n^*))$ **then**
      $n^* := n'$; $\bar{qv}_{T_0^s,T_1^s}(R(n^*)) := \bar{qv}_{T_0^s,T_1^s}(R(n'))$;
    end
  end
  $\mathbb{P}^{save} := \mathbb{P}^{save} \cup \mathbb{P}^{samples}$;
  if $|\mathbb{P}^{save}| > m^{save}$ **then**
    Keep only m^{save}-best personnel structures of $\mathbb{P}^{save}$;
  end
end
Calculate $\bar{qv}_{T_0^s,T_1^s}(R(n))$ for each $n \in \mathbb{P}^{save}$;
$n^* := \underset{n \in (\mathbb{P}^{save} \cup n^*)}{arg\ min} (\bar{qv}_{T_0^s,T_1^s}(R(n)))$;
Output: n^*;

Algorithm 1: The response surface methodology

RSM has a high computation time per-iteration because it needs to evaluate a number of personnel structures in one iteration. Moreover, as k increases, the accuracy of $F(n)$ is likely to decrease because more coefficients of $F(n)$ need to be estimated while the number of personnel structures in $P^{samples}$ is limited. Similar previous works have been tested with various problems with size ranging from 2 to 14 (Huang et al., 2006; Regis and Shoemaker, 2007). In order to deal with problems with larger size, we propose a simulated annealing algorithm that has a lower computation time per-iteration.

3.2 Simulated Annealing

Simulated annealing (SA) is a metaheuristics algorithm that mimics the behavior of annealing in a heat treatment process (for further information of SA, see e.g. Talbi (2009)). SA is selected since it only involves modifying one solution at one time. Population based metaheuristics need more computation time per iteration. Considering a restricted computation budget, it may not converge properly. SA is also selected because it has the ability to escape from local optima since RQS's solution space may

not be convex (Haspeslagh, 2012). Other single-solution based metaheuristics such as Tabu Search can also be considered.

The SA algorithm is shown in Algorithm 2. It starts with an initial personnel structure n^0. It keeps the best-so-far personnel structure n^* and explores the neighborhood of the current personnel structure n^{curr}. The algorithm iteratively explores a neighboring personnel structure n' while trying to improve. It always updates the current personnel structure when it founds a better personnel structure. To escape from local optima, it allows to accept a worse personnel structure with probability $e^{\frac{-\Delta E}{T}}$ as shown in Line 10. The algorithm is terminated upon meeting a stopping condition. Several stopping conditions can be considered including the stopping temperature, the maximum computation time, the maximum number of iterations, etc. In order to reduce the computation time per-iteration, two modifications explained in the beginning of Section 3 are employed in SA.

```
1  Input: cooling schedule; initial solution n^0;
2  Calculate q̄v_{T_0^s,T_1^s}(R(n^0));
3  n* := n^0; n^curr := n^0; P^save := {n^0};
4  q̄v_{T_0^s,T_1^s}(R(n*)) := q̄v_{T_0^s,T_1^s}(R(n^0)); qv_{T_0^s,T_1^s}(R(n^curr)) := q̄v_{T_0^s,T_1^s}(R(n^0));
5  T := T_max;
6  while stopping condition not met do
7      Generate a neighboring personnel structure n' from n^curr;
8      Calculate qv_{T_0^s,T_1^s}(R(n'));
9      ΔE := qv_{T_0^s,T_1^s}(R(n')) − qv_{T_0^s,T_1^s}(R(n^curr));
10     if ΔE < 0 ∨ Rand < e^{−ΔE/T} then
11         n^curr := n'; qv_{T_0^s,T_1^s}(R(n^curr)) := qv_{T_0^s,T_1^s}(R(n'));
12         P^save := P^save ∪ n';
13         if |P^save| > m^save then
14             Keep only m^save-best personnel structures of P^save;
15         end
16     end
17     if qv_{T_0^s,T_1^s}(R(n')) < q̄v_{T_0^s,T_1^s}(R(n*)) then
18         Calculate q̄v_{T_0^s,T_1^s}(R(n'));
19         if q̄v_{T_0^s,T_1^s}(R(n')) < q̄v_{T_0^s,T_1^s}(R(n*)) then
20             n* := n'; q̄v_{T_0^s,T_1^s}(R(n*)) := q̄v_{T_0^s,T_1^s}(R(n'));
21             n^curr := n'; qv_{T_0^s,T_1^s}(R(n^curr)) := qv_{T_0^s,T_1^s}(R(n'));
22         end
23     end
24     Update T;
25 end
26 Calculate q̄v_{T_0^s,T_1^s}(R(n)) for each n ∈ P^save;
27 n* := arg min_{n∈(P^save ∪ n*)} (q̄v_{T_0^s,T_1^s}(R(n)));
28 Output: n*;
```

Algorithm 2: The simulated annealing algorithm

The neighboring personnel structure n' is obtained by slightly modifying the current personnel structure n^{curr}. Let $n^{curr} = [\, n_1^{curr}\ n_2^{curr}\ ...\ n_k^{curr}\,]$. The method to

generate a neighboring personnel structure is randomly selected from the following operations:

- Increase n_i^{curr} for a random subgroup i while the others are kept fixed.
  Specifically, $n' = [\, n_1^{curr} \; ... \; n_i^{curr} + 1 \; ... \; n_k^{curr} \,]$, with i is selected randomly.
- Decrease n_i^{curr} for a random subgroup i while the others are kept fixed.
  Specifically, $n' = [\, n_1^{curr} \; ... \; n_i^{curr} - 1 \; ... \; n_k^{curr} \,]$, with i is selected randomly.
- Increase n_i^{curr} and decrease n_j^{curr} for two different random subgroups while the others are kept fixed.
  Specifically, $n'_i = n_i^{curr} + 1; n'_j = n_j^{curr} - 1; n_l = n_l^{curr}, \forall l \neq i, j$, with $i \neq j$, i and j are selected randomly.

Moreover, we restrict n' to the ones satisfying the constraints n_i^{LB} and n_i^{UB} for each subgroup i.

4 Experiments

Table 1 provides the data set to test the proposed algorithm and it is taken from Bilgin et al. (2012). The data set consists of six wards and they originate from real-world nurse rostering problems. For each ward, three scenarios are given. The normal scenario represents the most frequent condition in the ward, the overload scenario represents the condition in which the ward encounters higher coverage requirements than normal, and the absence scenario is the condition that corresponds to high absence requests. We use $\phi_{normal} = 0.6$, $\phi_{overload} = 0.2$ and $\phi_{absence} = 0.2$.

In our experiments, skill type and work hours category are used to divide the personnel into subgroups. Nurses can be divided based on their skill type, such as, trainee nurse, head nurse, regular nurse, etc. Work hours category represents the full time equivalent (FTE) such as 100%, 80%, etc. Columns m^{st} and m^{wh} represent the number of skill types and work hours categories in each ward. The value $k = m^{st}m^{wh}$ denotes the total number of subgroups, which varies between 8 and 20. The initial personnel structure for each ward is obtained from (Bilgin et al., 2012). We also provide the total FTE in the wards and the initial roster quality in the last two columns. The initial roster quality is obtained after applying the algorithm of Bilgin et al. (2012) to the problem instances.

Table 1 The data set for testing the proposed algorithm

No	Ward	Abbr.	m^{st}	m^{wh}	k	$\sum FTE$	$\bar{qv}_{T_0^s,T_1^s}(R(n))$
1	Emergency	Em	4	2	8	26.25	17,309
2	Geriatrics	Gr	2	4	8	17.06	11,423
3	Meal Preparations	MP	2	4	8	16.84	13,223
4	Psychiatry	Ps	3	3	9	16.50	8,733
5	Reception	Re	3	3	9	13.75	14,447
6	Palliative Care	PC	4	5	20	21.70	130,579

We determine n_i^{LB} and n_i^{UB} by allowing n_i to decrease or increase by maximum three. Specifically, $n_i^{LB} = max[0, n_i - 3]$ and $n_i^{UB} = n_i + 3$. The rostering algorithm from Bilgin et al. (2012) was executed for 30 seconds in order to obtain a solution for the rostering problem with $m^{opt} = 10$. T^{comp} is composed of T^{iter} denoting the

time spent on local search and T^{eval}, which corresponds to the evaluation time of $\bar{qv}_{T_0^s,T_1^s}(R(n))$ for $n \in \mathbb{P}^{save}$. Experiments are conducted with T^{iter} set to 10 hours for both SA and RSM. For each ward, both algorithms are run five times.

For SA, we set $T_{max} = 1000$, $T_{t+1} = 0.999T_t$, and m^{save} equal to 10. For RSM, we differentiate the parameter values in the first iteration from the others in order to obtain a good initial solution. For the first iteration, $m^{samples} = 100$, $m^{range} = 3$, $m^{save} = 10$, maximum computation for generating OD = 60 seconds, maximum computation for PSO = 30 seconds. For the other iterations, $m^{samples} = 30$, $m^{range} = 1$, $m^{save} = 10$, maximum computation fo generating OD = 30 seconds, maximum computation for PSO = 30 seconds. Furthermore, the method proposed by Xu (2002) is used to generate the optimal design (OD). If after four iterations RSM cannot obtain a better n^*, m^{range} is increased by one.

5 Results and Discussion

The results are shown in Table 2. It provides the average roster quality $\bar{qv}_{T_0^s,T_1^s}(R(n))$ for SA and RSM for five replications. Moreover, it provides the ratio between the $\bar{qv}_{T_0^s,T_1^s}(R(n))$ from RSM and SA to the one from the initial. The average of the total FTE is also provided both for RSM and SA.

Table 2 The results of the SA and the RSM

No	Ward	Initial		RSM			SA		
		$\bar{qv}_{T_0^s,T_1^s}(R(n))$	$\sum FTE$	$\bar{qv}_{T_0^s,T_1^s}(R(n))$	ratio	$\sum FTE$	$\bar{qv}_{T_0^s,T_1^s}(R(n))$	ratio	$\sum FTE$
1	Em	17,309	26.25	13,176	76.13%	29.20	13,301	76.85%	29.35
2	Gr	11,423	17.06	4,508	39.47%	15.05	3,728	32.64%	15.35
3	MP	13,223	16.84	4,436	33.54%	15.05	3,949	29.86%	15.33
4	Ps	8,733	16.50	5,182	59.34%	14.35	4,060	46.49%	14.20
5	Re	14,447	13.75	9,750	67.49%	14.65	10,212	70.69%	14.45
6	PC	130,579	21.70	50,583	38.74%	17.42	20,111	15.40%	15.85

As shown in Table 2, both RSM and SA are consistent that they can always improve the roster quality by providing similar total FTE values. In general, SA performs better than RSM except for the reception ward. When we compare the initial total FTEs with the ones resulted from RSM and SA, we can conclude that two wards need more resources. The total FTEs resulting from RSM and SA are higher than the initial total FTEs in the emergency and reception wards. For other wards, it seems better to decrease the resources.

We also observe that with a slightly different total FTE, a better roster quality can be achieved. By decreasing 1.5 FTE for geriatrics and meal preparation wards, the ratios of the roster quality become around 30%. It should be concluded that the total FTE in the initial data is already good enough. However, the personnel allocation into subgroups needs to be modified so that it matches the real requirements.

Generally, SA results in a slightly better margin than RSM except for the palliative care ward. This ward particularly has 20 subgroups which makes it harder to solve. SA consumes less computation time per iteration than RSM. This is shown in Table 3. SA has a larger number of iterations (m^{iter}) than RSM with a comparable computation time (T^{comp}). It can be suggested that the number of iterations for RSM is not adequate to solve a problem with a high number of subgroups.

Table 3 The Statistics of the proposed SA and RSM. T^{comp} consists of the

No	Ward	SA		RSM	
		m^{iter}	T^{comp} (hours)	m^{iter}	T^{comp} (hours)
1	Em	159.80	12.63	7.60	12.97
2	Gr	200.20	12.60	8.20	12.86
3	MP	227.00	12.55	8.60	13.18
4	Ps	232.00	12.59	8.20	13.08
5	Re	266.00	12.59	8.80	13.27
6	PC	190.60	12.70	7.80	13.41

6 Conclusions

We have provided an integrated staffing and rostering model, which can be used to obtain a personnel structure that is expected to facilitate good quality rosters. Specifically, our contribution is that the model suggests a personnel structure with a good roster quality for the medium term. Moreover, our model has taken into account demand and personnel behavior variabilities within the staffing period, which have not been considered in the literature. We also proposed a simulated annealing approach for solving the model. The experiments show that the algorithm performs well. In general, it performs slightly better than the response surface methodology which is the common method in the literature for solving expensive optimization problems.

Our work can be extended in many ways. Additional objectives such as personnel cost, personnel satisfaction rate, etc. can be incorporated in a new formulation of an integrated staffing and rostering. Moreover, constraints regarding the personnel structure can be incorporated in the future model, such as a fixed total number of personnel, a ratio between the number of personnel of two different subgroups, maximum total FTEs, etc. Additional aspects can also be considered, such as multi departments and multi staffing periods.

Bibliography

Jonathan F. Bard and Hadi W. Purnomo. Incremental changes in the workforce to accommodate changes in demand. *Health Care Management Science*, 9(1):71–85, February 2006. ISSN 1386-9620. doi: 10.1007/s10729-006-6281-y.

Jeroen Beliën, Brecht Cardoen, and Erik Demeulemeester. Improving workforce scheduling of aircraft line maintenance at Sabena Technics. *Interfaces*, 42(4):352–364, 2012. doi: 10.1287/inte.1110.0585.

Burak Bilgin, Patrick De Causmaecker, Benoît Rossie, and Greet Vanden Berghe. Local search neighbourhoods for dealing with a novel nurse rostering model. *Annals of Operations Research*, 194(1):33–57, 2012. doi: 10.1007/s10479-010-0804-0.

Peter Chan, Michael Hiroux, and Georges Weil. Strategic employee scheduling. In Edmund K. Burke and H. Rudová, editors, *PATAT 2006 - Proceedings of The 6th International Conference on the Practice and Theory of Automated Timetabling*, pages 157–166, 2006. ISBN 80-210-3726-1.

Patrick De Causmaecker and Greet Vanden Berghe. Relaxation of coverage constraints in hospital personnel rostering. In E. Burke and P. De Causmaecker, editors, *Practice and Theory of AutomatedTimetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 129–147. Springer, Berlin, Heidelberg, 2003.

Stefaan Haspeslagh. *Negotiation at the short and mid term level supported by analysis of nurse rostering problems*. PhD thesis, Computer Science, Kulak, 2012.

Deng Huang, Theodore T. Allen, William I. Notz, and Ning Zeng. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34(3):441–466, 2006.

Mark W. Isken and Walton M. Hancock. Tactical staff scheduling analysis for hospital ancillary units. *Journal of the Society for Health Systems*, 5(4):11–23, 1998.

Komarudin, Marie-Anne Guerry, Tim De Feyter, and Greet Vanden Berghe. The roster quality staffing problem - a methodology for improving the roster quality by modifying the personnel structure. *European Journal Of Operational Research*, 2013. doi: 10.1016/j.ejor.2013.05.009.

June H. Larrabee, Michelle A. Janney, C Lynne Ostrow, Mary Lynne Withrow, Gerard R. Hobbs Jr, and Christopher Burant. Predicting registered nurse job satisfaction and intent to leave. *Journal of Nursing Administration*, 33(5):271–283, 2003.

Yongjian Li, Jian Chen, and Xiaoqiang Cai. An integrated staff-sizing approach considering feasibility of scheduling decision. *Annals of Operations Research*, 155(1): 361–390, July 2007. ISSN 0254-5330. doi: 10.1007/s10479-007-0215-z.

Broos Maenhout and Mario Vanhoucke. An integrated nurse staffing and scheduling analysis for longer-term nursing staff allocation problems. *Omega*, 41(2):485–499, 2013.

Douglas C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, 2001.

Martin Mundschenk and Andreas Drexl. Workforce planning in the printing industry. *International Journal of Production Research*, 45(20):4849–4872, October 2007. ISSN 0020-7543. doi: 10.1080/00207540600813238.

Yasar A. Ozcan. *Quantitative methods in health care management: techniques and applications*. Jossey-bass, San Fransisco, second edition, 2009.

Rommel G. Regis and Christine A. Shoemaker. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19(4):497–509, 2007.

El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, 2009.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics With S*. Springer, 2002.

Ray Venkataraman and Michael J. Brusco. An integrated analysis of nurse staffing and scheduling policies. *Omega*, 24(1):57–71, February 1996. ISSN 03050483. doi: 10.1016/0305-0483(95)00051-8.

Hongquan Xu. An Algorithm for Constructing Orthogonal and Nearly-Orthogonal Arrays With Mixed Levels and Small Runs. *Technometrics*, 44(4):356–368, November 2002. ISSN 0040-1706. doi: 10.1198/004017002188618554.

MISTA 2013

A computational study of shifting bottleneck procedures for job shop total weighted tardiness scheduling problems

Bernd Scholz-Reiter • Yi Tan • Torsten Hildebrandt

Abstract Minimizing total weighted tardiness for job shops is a hard production scheduling problem. The shifting bottleneck (SB) procedure is considered to be one of the most effective heuristics to solve this problem. However, as a divide-and-conquer heuristic, the SB procedure consists of independently implementable components and for each of these components there are various solutions. The way to configure its components has a significant effect on the performance (i.e., scheduling quality) and efficiency (i.e., scheduling time) of the SB procedure. Aim of this paper is to give a guideline on appropriate configurations of the SB components. We review the component solutions in the literature and suggest improvements to some of them. Afterwards, we examine the performance as well as efficiency of the component solutions of the SB procedure by computational experiments. The experiments were conducted on static benchmark problem instances from the literature as well as on dynamic problem instances generated by dynamic job shop simulations. With the experiment results, we identify the effects of the different component solutions on the performance and efficiency of the SB procedure and suggest appropriate configurations regarding different problem sizes and scheduling time requirements.

1 Introduction

The job shop scheduling problem is widely used for modelling manufacturing processes and therefore has received great attention from scheduling research. Due to the increasing market competition, manufacturers currently pay more attention on due date related production objectives in order to fulfil the customers' demand on punctuality [1][2]. Consequently, the job shop scheduling problem with optimality criteria related to due date performance has become increasingly important.

In recent years a line of research has focused on minimizing the objective function total weighted tardiness (TWT) for job shops [3][4][5]. The weighted tardiness has been given an economical interpretation, referring to penalty costs paid to customers or the loss of goodwill of customers [1], which obviously have to be avoided as much as possible.

To solve this job shop TWT scheduling problem two types of techniques have been developed: exact methods and heuristics. Singer and Pinedo [3] describe a branch-and-bound

Bernd Scholz-Reiter
University of Bremen
E-mail: bsr@biba.uni-bremen.de

Yi Tan
University of Bremen
E-mail: tan@biba.uni-bremen.de

Torsten Hildebrandt
University of Bremen
E-mail: hil@biba.uni-bremen.de

method capable of solving 10×10 (jobs×machines) instances exactly. However, since the job shop TWT problem is strongly NP-Hard [3], the branch-and-bound method is incapable of solving larger instances. Among the heuristics methods, Vepsalainen and Morton [6] study a number of dispatching rules and conclude that the apparent tardiness cost (ATC) rule achieves the best results, while Pinedo and Singer [4] develop a Shifting Bottleneck (SB) procedure for minimizing the total weighted tardiness (SB-TWT).

In this paper we focus on the SB-TWT procedure, which is considered to be one of the most effective methods to solve this problem. The effectiveness of the SB procedure can be traced back to first development by Adams et al. [7] for the job shop makespan problem. They introduced the SB procedure as a divide-and-conquer heuristic, which decomposes a job shop problem into a number of single machine problems. At each iteration, a critical machine is identified and solved. The principle of a SB procedure is that a heavily loaded machine (bottleneck) will define almost the performance of the entire job shop and can be therefore optimized independently. The schedules of a lightly loaded machine should be obtained as a function of the bottleneck machine and have to be optimized optimized later [8]. Adams et al. [7] reported excellent performance and efficiency of their SB procedure on benchmark problem instances. For instance, the SB procedure solves the well-known 10x10 problem introduced by Muth and Thompson [9] optimally. A number of research studies have enhanced the SB procedure in different ways, for instance by adding delayed precedence constraints in the subproblems for preventing the generation of infeasible solutions [10, 11], by modifying the reoptimization or backtracking mechanisms [12, 13].

As a divide-and-conquer heuristic, the SB procedure consists of independently implementable components and for each of these components there are various solutions in literature. In order to examine the effect of the components and their solutions on performance (i.e., scheduling quality) and efficiency (i.e., scheduling time) in the SB procedure for the makespan problems, Holtsclaw and Uzsoy [14] present a computational study of different machine criticality measures and subproblem solution procedures. Ovacik and Uzsoy [15] examine the component solutions of the SB procedure by applying computational experiments on benchmark problem instances, while Demirkol et al. [16] analyse the time/quality trade-off of different component configuration of the SB procedure by means of computational experiments on an extensive set of randomly generated job shop problem instances. All of these studies confirm that the configuration of its components has a significant effect on the performance and efficiency of the SB procedure.

In the last 15 years, the SB procedure has been extended to solve the job shop TWT problems. Pinedo and Singer [4] first introduce their SB procedure (called SB-TWT) to solve this problem. They present two versions of the SB-TWT: a simple one with a ‘one-pass’ process and a complex one which uses a partial enumeration heuristic called backtracking. They show that the SB-TWT with backtracking solves 10×10 instances in most cases optimally. However, the SB-TWT backtracking version is incapable of solving instances larger than 30×10 [8]. This is because it is very sensitive to the number of jobs and machines due to the use of backtracking, which leads to the runtime of the procedure growing exponentially with the number of machines and/or jobs [8]. Therefore, a line of research [5, 17] focuses on the ‘one pass’ version of the SB-TWT because of its high efficiency. They enhance SB-TWT to solve larger and more realistic problem instances of complex job shop scenarios in the context of semiconductor wafer fabrication. According to these studies their SB solutions outperform other heuristic methods they studied. Another approach to overcome the inefficiency of the SB-TWT with backtracking is presented in Scholz-Reiter et al. [18]. They use Variable Neighbourhood Search (VNS) to replace the backtracking in two phases of the SB-TWT procedure. First, it is used as a high-quality sub-problem solution procedure. Second, it is used to improve the bottleneck planning sequence. They report that their SB version (called SB-VNS) can achieve at least the same schedule quality as the SB-TWT with backtracking, but requires a significantly lower runtime. This enables the SB procedure to better scale to larger and more realistic job shop problem instances.

Compared with the SB procedure for the makespan job shop problem, there has only been limited systematic effort to study the effect of the components and their different solutions on the performance and efficiency of the SB-TWT procedure. To the best of our knowledge, only Mönch and Zimmermann [19] present a simulation-based assessment of machine criticality measures of a SB-TWT procedure applying in a semi-conductor manufacturing environment, which is not representative for general job shops and only considers a single SB component.

This lack of a general and systematic effort highlights the need for a more detailed computational study of SB for job shop TWT problems. Therefore this paper aims to present a systematic analysis of the components and their various solutions of the SB procedure as well as a computational study of them regarding different problem sizes and chrematistics.

The paper is organized as follows: in the next section, we describe the job shop TWT problem. In Section 3, we present the SB procedure. In Section 4, we describe the components and their solutions of the SB procedure. In Section 5, we describe the environment and settings of our experiments. In Section 6, we present the results of the computational experiment on static problem instances from the literature with the focus on performance. In Section 7, we discuss experimental results on larger, realistic dynamic job shop instances with regard to the trade-off of performance and efficiency. The paper will be closed with conclusions and an outlook in Section 8.

2 Job shop total weighted tardiness problem

The job shop problem can be described as follows: a plant with m machines must process n jobs. Each job has a release date r_j and a predetermined routing that it has to follow through the different machines. The processing of job j on machine i is referred to as operation (i, j), with processing time $p_{i,j}$. Hence each job j consists of a predetermined sequence of n_j operation, where $n_j \leq m$ and no machine is visited by a job more than once.

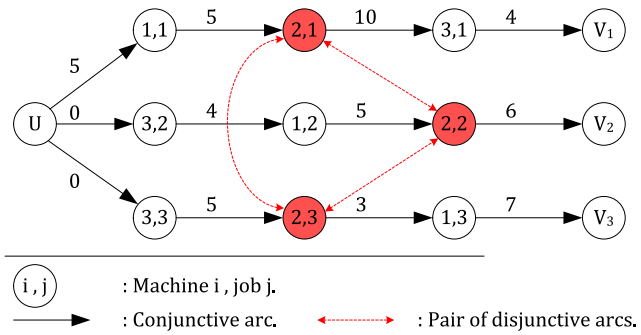


Fig. 1 Disjunctive graph for a job shop TWT problem [4].

Assume that each job j has a due date d_j and a weight w_j, let C_j denote the time that job j completes its last operation and define $L_j = C_j - d_j$ as the lateness of job j and $T_j = \max(L_j, 0)$ as the tardiness of job j, the objective function total weighted tardiness (TWT) is defined as $\sum w_j T_j := \sum_{j=1}^{n} w_j T_j$.

According to Pinedo [1] as well as Pinedo and Singer [4], the job shop TWT problem can be represented by a disjunctive graph $G = (N, A, B)$. Fig. 1 shows the disjunctive graph representation of a job shop problem instance with three jobs and three machines. The set of nodes N contains one element represented by a node (i, j) for each operation (i, j), one source node U and n sink nodes V_j. The set of conjunctive arcs $A = \{(i, j) \rightarrow (k, j)\}$ contains the arcs that connect the nodes representing each pair of consecutive operation (i, j) and (k, j) of job j. Each arc $(i, j) \rightarrow (k, j)$ has a length $|(i, j) \rightarrow (k, j)| = p_{i,j}$ and represents the constraint that operation (k, j) may be started no less than $p_{i,j}$ time units after operation (i, j) has been started.

The node ($n_{j,j}$) that represents the final operation of job j has an arc of length $p_{n_j,j}$ connecting to V_j. The source node U has n outgoing arcs, each one connects to the first operation of each job j, with length equal to the release date r_j. Let N_i denote the set of nodes corresponding to the operation processed on machine i. The set of disjunctive arcs $B = \{(i, j)\leftrightarrow(i, k)\}$ has, for every pair of nodes (i, j) and (i, k) in N_i, two arcs $(i, j)\leftrightarrow(i, k)$ going in opposite directions. Notice that, for the sake of clarity, only the disjunctive arcs related to machine 2 are depicted in the Fig. 1. The arc $(i, j) \rightarrow (i, k)$ has length $p_{i,j}$ and the arc $(i, k) \rightarrow (i, j)$ has length $p_{i,k}$. Each pair of arcs represents the fact that two operations cannot be processed simultaneously on the same machine. Fixing an arc in a certain direction corresponds to a decision as to which operation comes first. For instance, fixing arc $(2, 2) \rightarrow (2, 1)$ in Fig. 1 implies that operation $(2, 1)$ is processed after operation $(2, 2)$. Fixing all disjunctive arcs among operations to be processed on the same machine (e.g. the operations $(2, 1)$, $(2, 2)$ and $(2, 3)$ in Fig. 1) implies to fix the sequence in which the operations are processed on that machine.

A feasible schedule for the job shop problem corresponds to a selection $s(B)$ of one disjunctive arc from each pair $(i, j)\leftrightarrow(i, k)$ such that the resulting directed graph $G(N, A, s(B))$ is acyclic [1]. Let $L(v, v')$ denote the length of the longest path from node v to node v' in graph $G(N, A, s(B))$. If there is no such path, then $L(v, v')$ is not defined. The completion time C_j of job j is equal to $L(U, V_j)$. The values $L(U, v)$, v being any node, are computed using any algorithm solving longest path problem in a weighted directed acyclic graph. Pinedo and Singer [4] suggest to use the algorithm of Bellman [20], which has complexity $O(|N|)$ since each node has at most two incoming arcs. Through the completion time C_j represented by $L(U, V_j)$, the objective function TWT can be calculated. Thus, solving the job shop problem is reduced to finding a selection $s(B)$ of disjunctive arcs that minimizes the TWT.

3 Shifting bottleneck procedure

Shifting bottleneck procedure is a divide-and-conquer heuristic. It divides a job shop scheduling problem into m single machine scheduling problems and solves them successively. In this Section, we describe the shifting bottleneck procedure SB-TWT presented by Pinedo and Singer [4] as well as Pinedo [1]. The procedure can be briefly formulated as follows:

Phase 1: The job shop problem is decomposed into a number of single machine problems (subproblems) which have to be scheduled. Subproblems according to machines which are still not scheduled are formulated.

Phase 2: The subproblems are solved according to some specified 'subproblem solution procedure' (SSP).

Phase 3: A machine criticality measure is computed in order to rank all unscheduled machines in order of criticality. The schedule of the bottleneck, i.e., the most critical machine, is fixed.

Phase 4: The disjunctive graph representation is used to capture the interactions between the machines already scheduled and those not yet scheduled.

Phase 5: Those machines which already have been sequenced are rescheduled using the new information obtained in Phase 4. If all machines have been scheduled, stop. Otherwise go to *Phase 1*.

Let M denote the set of all m machines and M_0 denote the subset of machines, which have been scheduled. For each one of the machines in M_0 a sequence of operations has already been determined. That means a selection of disjunctive arcs already has been fixed for machines in M_0. At the beginning $M_0 = \varnothing$.

In *Phase 1*, a single machine problem is created for each machine i in $M - M_0$. In the first iteration, all machines are not scheduled. Therefore, m single machine problems are created. The disjunctive graph representation allows the calculation of release dates and due dates for

each operation on machine i, where such data depends on the schedule of other machines. The disjunctive arcs still not fixed are not taken into account for the longest path calculation in Graph G. The due date $r_{i,j}$ of an operation (i, j) is defined as $L(U, (i, j))$, the longest path from U to the note (i, j) in the disjunctive graph. A delay in the completion time of operation (i, j) may impact the tardiness of all n jobs. Therefore, an operation has n local due dates. The local due date $d^k_{i,j}$ of a operation (i, j) with respect to the due date of job k is defined as $\max(C_k, d_k) - L(V_k, (i, j)) + p_{i,j}$. If $L(V_k, (i, j))$ does not exist, then the local due date $d^k_{i,j}$ is infinity.

These data are used to define the $1 \mid r_{i,j}, \text{prec} \mid \sum w_k(\max T^k_{i,j})$ single machine scheduling problem for machine i. If operation (i, j) is completed after its local due date $d^k_{i,j}$, i.e., $C_{i,j} > d^k_{i,j}$, the tardiness of job k in the job shop increases by at least $C_{i,j} - d^k_{i,j}$. Let $T^k_{i,j} = \max(C_{i,j} - d^k_{i,j}, 0)$ denote the tardiness of operation (i, j) with respect to the due date of job k. Because all operations (i, j) assigned to machine i have to be scheduled, the tardiness of job k increases by at least $\max_{(i,j)\in N_i} T^k_{i,j}$ and the total increase in the job shop objective function will be at least $\sum_{k=1}^{n} w_k(\max_{(i,j)\in N_i} T^k_{i,j})$. The constraint 'prec' considers the delay precedence constraints of operations on machine i, which is crucial for avoiding cycles in the disjunctive graph G [10, 11]. There is a prec $(i, j) \rightarrow (i, k)$, if and only if a path from operation (i, j) to operation (i, k) exists and its length is given by the longest path $L((i, j), (i, k))$ [4]. These constraints can be calculated by using an extension of the algorithm by Bellman [20]. This resulting problem denoted by $1 \mid r_{i,j}, \text{prec} \mid \sum w_k(\max T^k_{i,j})$ for machine i is strongly NP-hard, since it is a generalization of the problem of minimizing the maximum tardiness in a single-machine $1 \mid r_j \mid T_{max}$ [4].

In *Phase 2*, the resulting subproblems are solved by a SSP. Different SSPs to solve the $1 \mid r_{i,j}, \text{prec} \mid \sum w_k(\max T^k_{i,j})$ problem will be discussed in detail in Section 4. The solution for the optimization of subproblem i corresponds to a sequence of operations processed on machine i, and the value of this solution provides the estimate $\sum w_k(\max T^k_{i,j})$ for the increase in the job shop's total weighted tardiness caused by this machine schedule.

In *Phase 3*, it has to be determined which machine in $M - M_0$ has to be planed as next in set M_0. The solutions of the subproblems according to machines in $M - M_0$ are evaluated by a machine criticality measure (MCM). The value of the objective function $\sum w_k(\max T^k_{i,j})$ of the solutions is one possible MCM. Nevertheless, there are still other more complex MCMs, which will be discussed in detail in Section 4. The machine with the most criticality value is considered as the next bottleneck. Its schedule (the operation sequence) is inserted into the job shop schedule, by fixing in the disjunctive graph the direction of those disjunctive arcs that represent such a machine schedule. In addition, the selected machine is moved into the M_0.

In *Phase 4*, the release dates and local due dates of all operations in the disjunctive graph G have to be updated, after that a new critical machine has been scheduled and its corresponding disjunctive arcs have been fixed in the graph G.

The *Phases* 1-4 can be performed in a 'one-pass' process which schedules the machine with the highest criticality value for each iteration until all the machines are scheduled, then stop.

In *Phase 5*, an additional reoptimization process can enhance the 'one-pass' mechanism by optimizing the schedules of those machines in M_0 which were scheduled in previous iterations, each time when a new machine is scheduled. The reoptimization process can be briefly described as follows: Removing the schedule of a machine i in M_0 by removing its corresponding disjunctive arcs in the graph G, while fixing the disjunctive arcs of other machines in M_0; Updating the release dates and local due dates of all operations in the graph through the method described in *Phase 1*; Solving the subproblem of the machine i again with the SSP. Because there is now more information about the schedules of the other machines which have been scheduled, it is expected that the new schedule for machine i should be more

precise than the old one. The diverse reoptimization procedures will be still discussed in Section 4.

Moreover, further mechanisms can be used to optimize the machine planning sequence, in which the machines in M_0 are scheduled, to enhance the schedule quality. These mechanisms will be discussed in the following section.

4 Components of the SB procedure

In this section we will discuss the components of the SB procedure described in the previous section in detail and review the solutions for each component considered in this paper.

4.1 Subproblem solution procedure

4.1.1 ATC based priority rules:

Pinedo and Singer [4] developed the following priority rule to schedule the operations on a single machine:

$$I_{ij}(t) = \sum_{k=1}^{n} \frac{w_k}{p_{i,j}} \exp\left(-\frac{(d_{ij}^k - p_{i,j} + (r_{i,j} - t))^+}{k\overline{p}}\right) \quad (1)$$

Where the parameter $\overline{p}$ denotes the average processing time of all operations and k is a scaling parameter that has to be determined empirically by computational experiments. However, if we consider the rolling time horizon method for dynamic job shops, we determine that this rule depends on the chosen time horizon and therefore may have unexpected effects on the dynamic job shop scheduling. (In this paper, a dynamic job shop means a job shop with non-zero release dates of jobs and machines.)

Let L is the length of a time horizon. The same operation in the next time horizon (the same situation is shifted by L time units) has the following priority index given by the rule:

$$I_{ij}(t) = \sum_{k=1}^{n} \frac{w_k}{p_{i,j}} \exp\left(-\frac{((d_{ij}^k + L) - p_{i,j} + ((r_{i,j} + L) - (t + L)))^+}{K\overline{p}}\right)$$

$$= \quad (2)$$

$$I_{ij}(t) = \sum_{k=1}^{n} \frac{w_k}{p_{i,j}} \exp\left(-\frac{(d_{ij}^k + L - p_{i,j} + (r_{i,j} - t))^+}{K\overline{p}}\right)$$

The index (2) does not equal the (1). It turns out that the priority index (1) depends on L. Therefore this priority rule is not time invariant and can even cause different job sequences depending on different values of L. Furthermore, the formulation causes numerical problems if L becomes very large and thus the exponential term approaches zero.

The concrete effect of the dependence of the time horizon of this rule will be examined in Section 6. In order to remove this dependence, we investigated the Apparent Tardiness Cost (ATC) rule [6] , which is the basis of the Pinedo and Singer rule, and developed three further rules, which are independent of L (see Table 1).

4.1.4 Backtracking as SSP

Pinedo and Singer [4] present a partial enumeration heuristic with backtracking to solve the subproblems in the SB procedure. For a single machine problem, they use the PINEDORULE to prioritize the unscheduled operations. All operations are ordered in a priority queue. With backtracking, they do not only select the operation with highest priority as the next operation,

but consider the first α (α is the backtracking aperture and α >1) operations in the front of the priority queue. Considering a decision or search tree, the backtracking heuristic appends these α operations as children to the last decision (the last scheduled operation). Each of these α operations will have further α children as its possible successors in the decision tree. In total, $\alpha^{(n-\alpha)}\alpha!$ solutions (represented by operation permutations) are considered by the backtracking, where n is the number of operations of the single machine problem. The solution with the best objective value will be used as the solution for the single machine subproblem. Hence, this backtracking heuristic has an exponential runtime of $O(\alpha^{(n-\alpha)}\alpha!)$ [18]. Pinedo and Singer [4] use backtracking with α=10 as SSP to solve the 10x10 job shop problem instances, i.e., single machine problems are always solved optimally. Thus PINEDORULE is only used to provide a pre-ordering of jobs which does not affect final solution quality but only the effectiveness of the branch-and-bound algorithm which is used to reduce computational time of their backtracking approach.

Table 1 Dispatching rules as SSP for dynamic job shops

Rule	Definition	Comment
PINEDORULE	$I_{ij}(t)=\sum_{k=1}^{n}\frac{w_k}{p_{i,j}}\exp(-\frac{(d_{ij}^k-p_{i,j}+(r_{i,j}-t))^+}{k\bar{p}})$	Not time invariant; can cause numerical problems
sumATC	$I_{ij}(t)=\sum_{k=1}^{n}\frac{w_k}{p_{i,j}}\exp(-\frac{(d_{ij}^k-p_{i,j}-t)^+}{k\bar{p}})$	Sum of basic ATC rule; no consideration of release date
sumATC_RP	$I_{ij}(t)=\sum_{k=1}^{n}\frac{w_k}{p_{i,j}}\exp(-\frac{(d_{ij}^k-p_{i,j}-t+(r_{i,j}-t)^+)^+}{k\bar{p}})$	Same as above but additionally penalizing jobs released in the future
sumATC_R_2T	$I_{ij}(t)=\sum_{k=1}^{n}\frac{w_k}{p_{i,j}}\exp(-\frac{(d_{ij}^k-p_{i,j}+r_{i,j}-2t)^+}{k\bar{p}})$	Same as before but without inner max-term

4.1.5 Variable neighbourhood search as SSP

Geiger [21] presents a variable neighbourhood search (VNS) heuristic to solve the $1||\sum w_jT_j$ single machine TWT problem. He reports an excellent performance and efficiency of the VNS. Scholz-Reiter et al. [18] extended the VNS heuristic of Geiger [21] from $1||\sum w_jT_j$ to the $1|r_j, \text{prec}|\sum w_k(\max T_j^k)$ problem, adding the ability to account for release dates for jobs and machines as well as precedence constraints of jobs. Multiple due dates of operations and the objective function are also modified as appropriate. They report a promising performance of using the VNS heuristic as SSP for the SB procedure. The runtime of $O(n^2)$ of the VNS enables the SSP to better scale to larger and more realistic single machine problem instances.

For the sake of completeness, we outline here the VNS briefly. VNS [22, 23] attempts to improve an initial solution by finding better solutions in some neighbourhood around it. If such an improvement is found, it becomes the new starting point. A further improvement is

attempted, now considering the solutions in the neighbourhood of this new starting point. VNS extends this basic hill-climbing behaviour by not merely using a single, but multiple neighbourhoods. A certain neighbourhood is used until no further improvements can be found, i.e., a local optimum has been reached. Afterwards, further progress is attempted by using a different neighbourhood to potentially move away from the previous local optimum. The neighbourhoods used in Scholz-Reiter et al. [18] are illustrated in Fig. 2. An initial permutation, i.e., operation sequence, is modified by first considering all backward shifts of an operation, afterwards forward shifts, and finally pairwise exchanges of operations until no further improvements are found.

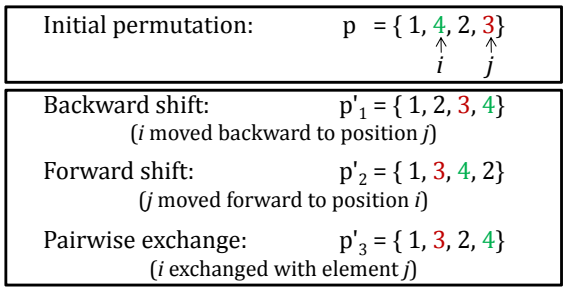


Fig. 2 Illustration of VNS neighbourhoods used by [18]

4.2 Bottleneck selection and machine criticality measurement (MCM)

A measure for the criticality of machines can be computed in a number of ways. In this work, we consider three MCMs, which are listed in Table 2. The PINEDO_MCM is described by [1] as follows: by selecting the disjunctive arcs implied by the schedule for machine *i*, one can easily compute the new disjunctive graph the new (overall) completion times of all n jobs, say C_k'' . Clearly, $C_k'' \geq C_k'$. *k_MCM* is a scaling parameter and has to be determined.

Table 2 Machine criticality measurements

MCM	Definition
TWT	The TWT value of the job shop, when the schedule of the machine is fixed into the disjunctive graphic.
Subproblem solution	The returned objective value of the subproblem solution of the machine.
PINEDO_MCM	$\sum_{k=1}^{n} w_k (C_k'' - C_k') \exp\left(-\frac{(d_k - C_k'')^+}{k_MCM}\right)$

4.3 Reoptimization

An important component of SB is the reoptimization procedure. The reoptimization procedure can be described as follows [7, 4, 16]: For each scheduled machine $k \in M_0$, we delete its job sequence and solve the single machine problem for machine *k* with the SSP again while keeping the sequences on all machines that are already scheduled fixed. If an improvement is found after this step, the process is repeated. During the first reoptimization cycle, the machines are considered in the order in which they were introduced into the solution. After the first reoptimization cycle, the machines in M_0 are reordered in decreasing order of their MCM values at the end of each reoptimization cycle. A more detailed description of the reoptimization procedure can be found in Adams et al. [7]. Furthermore, the number of

reoptimization cycles can be adjusted. Adams et al. [7] and Demirkol et al. [16] limit the number of reoptimization, because it is computationally intensive. However, Applegate et al. [12] suggest applying the reoptimization until there is no improvement found.

Finally, upon completion of the reoptimization procedure for a given M_0, Adams et al. [7] found it useful to repeat the procedure after temporarily removing from the problem the last δ noncritical machines from the solution for the remaining machines. They suggest to take δ to be the minimum of $|M_0|^{1/2}$ and the number of noncritical machines in M_0. At the end the non-critical machines are reintroduced into the solution and the reoptimization process is repeated. Applegate et al. [12] call this additional reoptimization process 'Twiddle' in their SB implementation.

It still remains the question, how we define the noncritical machines. For the job shop makespan problem, Adams et al. [7] defines that a machine is noncritical if no arc corresponding to an operation on the machine is on the critical path in the disjunctive graph. In other words, removing the solution of this machine does not decrease the makespan of the overall job shop, i.e., $\Delta Makespan = 0$. For the job shop TWT problem, we can similarly define a noncritical machine to be a machine with $\Delta TWT = 0$. However, instead of one critical path in the disjunctive graph for job shop makspan problem, there are *n* critical paths in the disjunctive graph for job shop TWT problem. It is usually difficult to find any machine with $\Delta TWT = 0$. Therefore, we suggest another definition for noncritical machines. For each iteration, the last $|M_0|^{1/2}$ machines with lowest MCM will be defined as noncritical. We denote this noncritical machine definition as *minMCM*.

4.4 Optimization of the machine planning sequence

The SB procedure repeatedly selects the next bottleneck among all unscheduled machines, until all machines have been scheduled. In total there are m! possible planning orders of machines. The machine permutations control the SB's progress. Similar to the single machine SSP, Pinedo and Singer [4] suggest a backtracking approach to search for an appropriate machine order: at each iteration, a certain number (denoted by β) of the most critical unscheduled machines are considered by the backtracking as the next machine to be scheduled. Hence, this backtracking approach considers an amount of $\beta^{(m-\beta)}\beta!$ different machine permutations. Pinedo and Singer use $\beta = 2$ and 3 for their SB-TWT procedure.

In order to avoid the exponential growth of runtime of backtracking with an increasing number of machines, Scholz-Reiter et al. [18] suggest to use VNS to improve an initial machine permutation, which is generated by a 'one pass' version of the SB procedure (i.e., β = 1). The VNS procedure considers an amount of $O(m^2)$ machine permutations. When used to optimize the machine planning order, we use the same neighbourhoods as shown in Fig. 2.

For this paper we could further enhance the efficiency of VNS by using a new caching scheme. Evaluating a certain machine planning sequence can be very time-consuming, as it might involve the SSP and update the disjunctive graph very frequently. To save computation time we store partial schedules in a cache data structure. If a new machine sequence has to be evaluated, we look up the entry with the longest common prefix between cache entries and the new machine permutation.

As an example consider a problem with 5 machines. Say, the one-pass version of SB (β=1) leads to a planning sequence (3,1,2,4,5). During this process, the partial schedules for the partial machine sequences (3), (3,1), and (3,1,2) are stored in the cache. If now a permutation (3,1,2,5,4) has to be evaluated (e.g., created by exchanging the fourth and fifth element of the initial permutation) we look up the partial schedule of the longest common prefix of both sequences, which in this example is the (3,1,2). We restore the associated partial schedule and complete the schedule by planning machine 5 and 4. In this process, new schedules for the planning sequences (3,1,2,5) and (3,1,2,5,4) are added to the cache.

Effectiveness of the cache depends on the average length of the common prefixes. If this prefix has zero length, because a new machine is on the first position in the permutation, there will be obviously no savings. In our experiments we could reduce run time by about 30% when solving the 10x10 instances with caching activated. All experiments using VNS to optimize the machine planning sequence reported in the remainder of this paper are conducted with caching enabled.

5 Computational experiments

The components of the SB procedure and their solutions described in Section 4 were implemented in Java. The computational experiments were run on a workstation with Windows Server 2003 using a 3.0 GHz Intel Xeon CPU and 32 GB of memory.

Two types of problem instances were used for the experiments. The first type of instances is the set of 66 small 10x10 (jobs × machines) benchmark instances used by Pinedo and Singer [4]. The results of these experiments are described in Section 6.

The second type of problem instances is a set of 40 large, dynamic, stochastic job shop instances. To generate these instances, we follow the procedure from Holthaus and Rajendran [24] to generate dynamic job shop instances. A system consisting of 20 machines and continuously processing jobs for a duration of one month was simulated. Each job entering the system is assigned a random routing, i.e., machine order is random and no machine is visited more than once. Processing times are drawn from a uniform discrete distribution ranging from 1 to 49 minutes. Job arrival is a Poisson process, i.e., inter-arrival times are exponentially distributed. The mean of this distribution is chosen to reach a desired utilization level on all machines. For the dynamic experiments in this paper, we chose a (mean) machine utilization of 90 %. Due dates of jobs are set with a due date tightness factor f = 4, which defines the due date of job j as its release date plus the sum of the processing times of its operation multiplied by f as follow:

$$d_j = r_j + \left\lfloor f(\sum_{i=1}^{m} p_{i,j}) \right\rfloor \tag{3}$$

Simulating this system for 30 days creates problem instances containing 1500-1700 jobs. Solving such instances directly is clearly infeasible even with the most effective version of SB. Therefore instances are further decomposed in smaller problem instances using a rolling time horizon with a 36 hours horizon. Simulating the shop using the ATC dispatching rule, we generate smaller problem instances for individual horizons which are used to assess the performance of various SB settings. Following this procedure we save one randomly chosen horizon from each large problem instance. These smaller instances consist of 20 machines and a varying number of 100 to 150 jobs. This way of generating dynamic instances should result in more realistic distributions for due dates, release dates and weights of jobs. Results on these dynamic problem instances are described in Section 7.

Table 3 Criteria for experiments

Criterion	Definition
Avg. Gap	$= \frac{\sum_{I \in S}(TWT(H,I) - BEST(I))}{\#S}$
Total Gap	$= \frac{\sum_{I \in S} TWT(H,I)}{\sum_{I \in S} BEST(I)}$
Opt. Gap	$= \frac{\sum_{I \in S} TWT(H,I)}{\sum_{I \in S} Opt(I)}$
#BEST	Number of times variant H finds the BEST

	solution over all instances in set *S*
Avg. Time	Mean computation time in seconds over all instances in set *S*

For evaluation of the solution variants of the SB procedure, we use diverse criteria. Let $TWT(H, I)$ be the TWT value obtained by SB variant H on a given instance I, and $BEST(I) = \min_H \{TWT(H, I)\}$. In addition, let $Opt(I)$ be the optimal TWT value of the instance I, when its optimum is known. We define a problem set S, on which the SB variants are examined. Hence, we can define the criteria in Table 3:

The criterion 'Avg. Gap' is statistically more accurate than the 'Total Gap' and 'Opt. Gap'. However, this criterion has difficulties, when $BEST(I) = 0$ for some instances in set S. These instances have to be excluded from the calculation of the 'Avg. Gap'. Therefore, we use 'Opt. Gap' to evaluate the SB variants for the 10x10 instances, because the optimums of these instances have been provided by Singer and Pinedo [3]. For the large problem instances, we use the 'Total Gap' as suggested by Demirkol et al. [16], when $BEST(I) = 0$ occurs.

Pinedo and Singer [4] denote their SB backtracking version by SB(α, β) with backtracking aperture $\alpha>1$ for single machine subproblems and backtracking aperture $\beta>1$ for machine planning sequence. Scholz-Reiter et al. [18] denote their SB-VNS versions by SB(VNS, 1) and SB(VNS, VNS). The SB(VNS, 1) only uses VNS for solving the single machine problems and SB(VNS, VNS) uses VNS for optimizing both single machine problems and the machine planning sequence. For our experiments, we use the same notation to denote the SB variants.

6 Results for static problem instances

6.1 Subproblem solution procedure (SSP)

6.1.1 Determine k for ATC based priority rules

Vepsalainen and Morton [6] state that the k of the original ATC rule for job shop problem is usually within [1.5, 4.5]. Pinedo and Singer [4] demonstrate that the k of their rule for single machine subproblems should be determined by computational experiments and use k=0.1 for the example in their paper. Lee and Pinedo [25] test k within [0.2, 6.4] and show that the k value of ATC for parallel machine scheduling problems depends on the characteristics of problem instances, especially on the number of jobs and machines and due dates of jobs.

In this experiment step, we will examine the influence of the k value on the performance of all ATC-based rules for the subproblems in order to identify an appropriate k value for further experiments. To exclusively examine the effect of k on the SB procedure, we use the simplest 'one pass' SB procedure in this experiment step. It means the machine criticality measure is the TWT value of the subproblem. No reoptimization and other advanced mechanisms are used.

We apply a rough test for parameter k. The ATC-based rules are applied for different values of k = (0.0001, 0.01, 0.1, 0.6, 1.1, …, 6.6, 8, 10, 20, 40, 60, 80, 100, 10000). Step size for k in the interval [0.1, 6.6] is 0.5.

Fig. 3 shows the 'Avg. Gap' of the dispatching rules with different k values on the whole 66 instances. The ATC rule is constantly worse than the other rules, independent of the k values. The best results of rules are achieved with values for k in the interval [0.1, 1.6]. Therefore, we apply a fine test for k in the interval [0.1, 1.6] with a step size of 0.1 as shown in Fig. 4. Table 4 shows 'Avg. Gap' of the best k and the mean of 'Avg. Gap' of the best k interval for each dispatching rules.

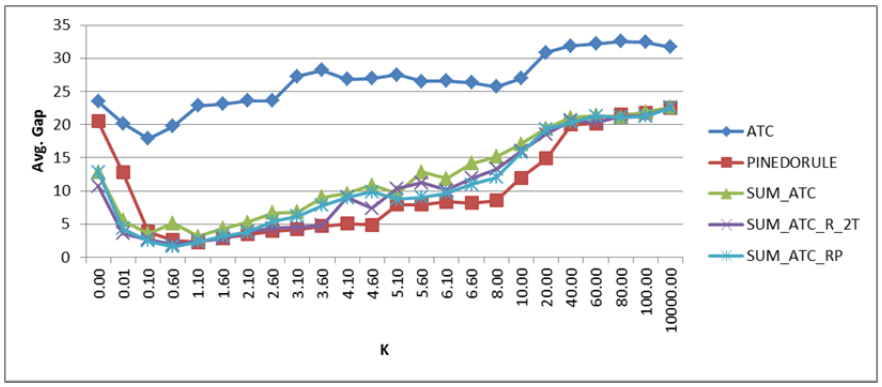


Fig. 3. Rough test of the k for ATC-based priority dispatching rules

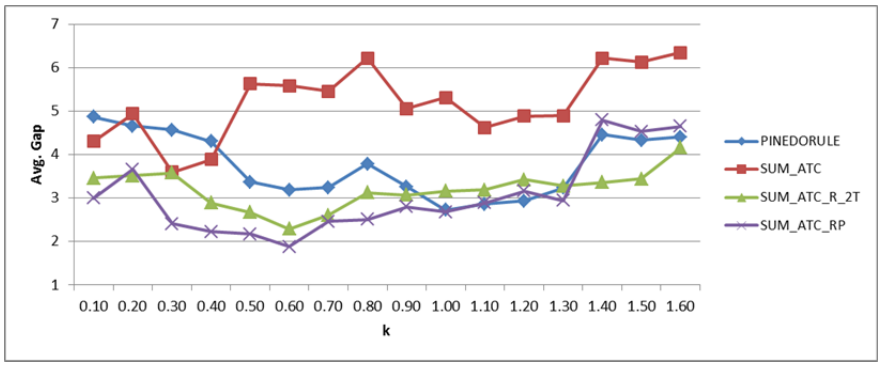


Fig. 4. Fine test of k for the set of 10x10 problem instances

Table 4 The best k and the best value interval of k for ATC-based subproblem solver

PINEDORULE		SUM_ATC_RP		SUM_ATC_R_2T	
Best *k*	Avg. Gap	Best *k*	Avg. Gap	Best *k*	Avg. Gap
1.0	2.72	0.6	1.87	0.6	2.28
avg. [0.3, 1.3]	3.40	avg. [0.3, 1.3]	2.55	avg. [0.3, 1.3]	3.02

Instead of trying to identify a single best value to be used for all single machine problems, *k* can also be determined dynamically by trying different values for each single machine problem and choosing the best solution obtained. To test this approach, we apply the dispatching rules with 11 different *k*-values within [0.3, 1.3]. For each single machine problem, a rule solves it with each *k*-value and selects the schedule with the best solution from the 11 schedules obtained. Table 5 shows the result. The 'Avg. Gap' of the rules with a fixed *k* was changed in comparison with Table 4, because for some instances new best solutions were generated by rules with dynamic *k*. The performance of the PINEDORULE was enhanced considerably, while the performance of the other two rules by using dynamic *k* is close to the one obtained with the best fixed *k*.

Table 5 Dynamic *k* for ATC-based SSP.

PINEDORULE		SUM_ATC_RP		SUM_ATC_R_2T	
k	Avg. Gap	*k*	Avg. Gap	*k*	Avg. Gap
1.0	2.71	0.6	1.91	0.6	2.34
Dynamic	2.27	Dynamic	2.12	Dynamic	2.35

6.1.2 Backtracking and Variable Neighbourhood Search

We apply the SB(VNS,1), SB(10,1) and SB(2,1) on the problem instances. Different dispatching rules are used to generate the initial solution for the VNS procedure. All dispatching rules use $k = 0.6$. As shown in Table 6, the initial solution is crucial for the VNS. An initial solution created by an ineffective method (for instance, a job permutation randomly generated by the rule 'RANDOM') affects not only the performance but also the efficiency of the VNS procedure. VNS needs more iterations to move to a high quality solution and is more likely to be trapped in a local optimum along the way. The SUM_ATC_RP and the PINEDORULE are the most promising methods to generate initial solutions for VNS. SUM_ATC_RP has a better 'Avg. Gap', while the other two criteria 'Opt. Gap' and #BEST of PINEDORULE are better.

Table 6 Test results of backtracking and VNS for subproblems

SB(VNS,1)	RANDOM	ATC	SUM_ATC	SUM_ATC_R_2T	SUM_ATC_RP	PINEDORULE
Avg. Gap	6.17	3.78	2.64	2.41	2.05	2.20
Opt. Gap	4.10	3.44	3.31	3.24	3.22	3.04
#BEST	3	5	4	8	5	9
Avg. Time	0.21	0.03	0.02	0.03	0.02	0.03
SB(10,1)						
Avg. Gap		3.57	2.46	2.25	2.00	2.36
Opt. Gap		3.18	3.27	3.08	3.16	3.07
#BEST		6	7	10	7	10
Time		0.52	0.57	0.56	0.55	0.55
SB(2,1)						
Avg. Gap		15.65	21.19	21.56	27.06	22.60
Opt. Gap		12.69	14.83	15.50	16.10	14.58
#BEST		0	0	0	0	0
Avg. Time		0.03	0.03	0.03	0.03	0.03

SB(10,1) applies already a full enumeration for the test instances with 10 jobs for each machine, i.e., single machine problems are always solved optimally. As shown in the results, however, the dispatching rule used to lead the backtracking also influences the performance of the SB(10,1). That is because a single machine problem can have more than one optimal solution. Especially in the first iterations of the SB procedure, subproblems usually have optimal solutions with TWT=0. An effective dispatching rule can choose a good solution among all optimal solutions, which might be better for future iterations of the SB procedure.

SB(VNS,1) has the most promising performance/efficiency trade-off for the single machine subproblems. Its performance is close to the SB(10,1). It requires only the same computational effort as the SB(2,1), however.

6.2 Bottleneck selection and machine criticality measurement (MCM)

We test the effects of the three MCMs described in Section 4.2 on the performance as well as efficiency for the SB procedure. In order to compare the PINEDO_MCM with the other two, we have to determine its scaling parameter *k_MCM* first. For the sake of brevity, we only use the PINEDORULE with k=0.6 as well as SB(VNS,1) as two subproblem solution procedures for this experiment.

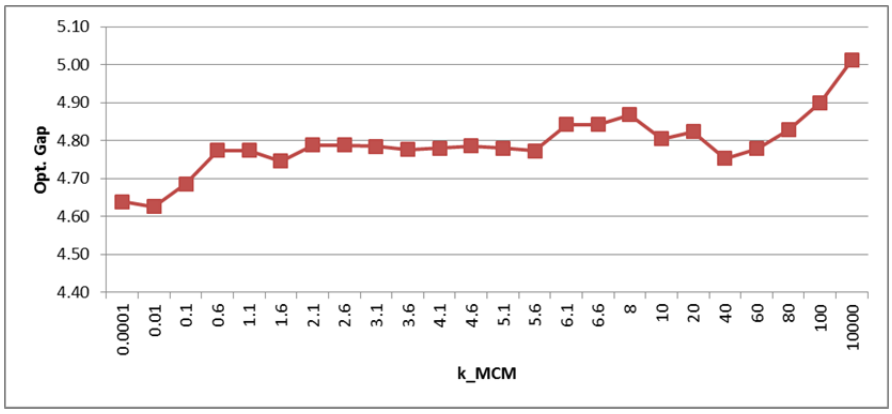


Fig. 5 Results for k_MCM generated by PINEDORULE

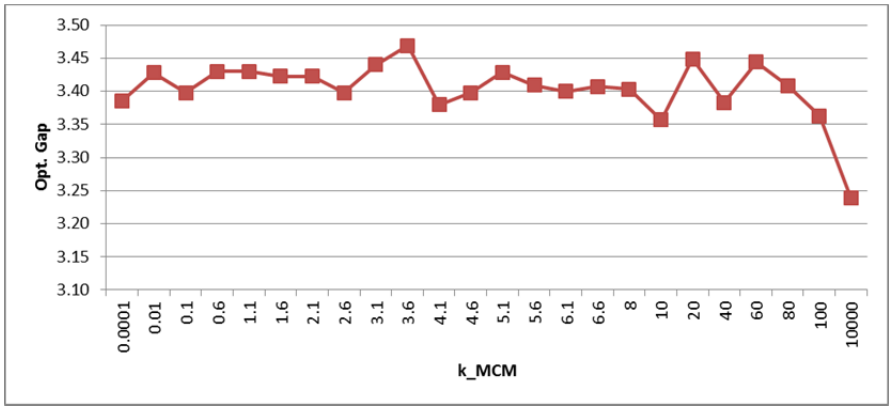


Fig. 6 Results for k_MCM generated by SB(VNS,1) using the initial solution generated by PINEDORULE

Similar to the test of the scaling parameter k for subproblem solution procedures, we test the *k_MCM* with the following values: *k_MCM* = (0.0001, 0.01, 0.1, 0.6, 1.1, …, 6.6, 8, 10, 20, 40, 60, 80, 100, 10000). In the interval [0.1, 6.6] the increase of *k_MCM* is 0.5. As shown in the Fig. 5 and Fig. 6, the two results tend to go to an opposite direction. While the best performance is achieved by *k_MCM* = 0.01 in Fig. 5, the best result is reported by *k_MCM* = 10000 in Fig. 6. In addition, the results do not converge to any value area of *k_MCM*. However, the 'Opt. Gap' values show little sensitivity to changing values of *k_MCM*. For instance, all 'Opt. Gap' in Fig. 6 are between 3.25 and 3.5. Hence, we choose *k_MCM* = 4.1 as a value in the middle of the value range for further experiments.

The next experiment examines the performance of the MCMs. Table 7 reports the 'Opt. Gap' of the three MCMs. The MCMs 'Subproblem Solution' and 'TWT' have the same performance and outperform the 'PINEDO_MCM'.

Table 7 Comparison of effect of MCMs on the performance of the SB measured by 'Opt. Gap'

	PINEDORULE	SB(VNS,1)
PINEDO_MCM	4.78	3.37
Subproblem Solution	4.63	3.03
TWT	4.63	3.03

6.3 Reoptimization

We examine six SB variants regarding reoptimization and twiddle, which are listed in Table 8. We do not limit the number of reoptimization cycles for each reoptimization. This means reoptimization is performed until no better solution can be found. We also consider the frequency of using reoptimization: for each iteration of the SB procedure, when a bottleneck machine is identified, or only for the last iteration of the SB procedure, when all machines are already planned.

For every combination of reoptimization and twiddle, the uncritical machine definition 'min TWT' always outperforms the 'ΔTWT=0'. The use of normal reoptimization for each iteration in the SB procedure enhances the SB performance most significantly. It reduces the 'Opt. Gap' of the SB procedure using PINEDORULE from 4.63 to 2.59, a reduction by 44%. Therefore, the reoptimization should be used as a mandatory component. Furthermore, twiddle improves the performance.

With reoptimization and twiddle for each iteration, the 'Opt. Gap' is further reduced from 2.49 to 2.15. Looking at the figures for SB(VNS,1) there is an improvement from 3.03 to 1.98 to 1.58, reducing the gap to the optimal solutions by about 48%. Hence, for the small problem instances, reoptimization and twiddle should be used for each iteration of the SB procedure.

Table 8 Reoptimization and twiddle

	No Reopt., No Twiddle		Reopt., No Twiddle, Each iteration		Reopt., Twiddle, (ΔTWT=0), last iteration		Reopt., Twiddle (min TWT), last iteration		Reopt., Twiddle (ΔTWT=0), Each iteration		Reopt., Twiddle (min TWT), Each iteration	
	Opt. Gap	Time [s]	Opt. Gap	Time [s]	Opt. Gap	Time [s]	Opt. Gap	Time [s]	Opt. Gap	Time [s]	Opt. Gap	Time [s]
PINEDO-RULE	4.63	0.03	2.59	0.13	2.81	0.1	2.74	0.1	2.22	0.33	2.15	0.33
SB(VNS,1)	3.03	0.03	1.98	0.15	2.04	0.12	2.02	0.12	1.68	0.38	1.58	0.4

6.4 Machine planning order

In this phase, we test the backtracking and VNS variants of the SB procedure. Table 9 shows the SB variants we tested and their results. The notation description of the SB variants is given in the Section 5. The SB(one pass) uses the PINEDORULE as SSP and applies no reoptimization and no further mechanism. For the other SB variants, the reoptimization and twiddle were applied. The SB(VNS, VNS) has the best performance among all variants we tested. It deviates only 0.07 from the optimum and achieved 50 from total 66 best solutions. The SB(10, 3) has a similar deviation to the optimum like SB(VNS, VNS), but achieved only 35 best solutions.

Table 9 Performance and computational time of SB variants

	SB (one pass)	Best Rule	SB (1,1)	SB (VNS, 1)	SB (1,VNS)	SB (10, 2)	SB (10, 3)	SB (VNS,VNS)
Opt. Gap	4.63	2.48	2.16	1.55	1.34	1.20	1.08	1.07
#BEST	1	1	1	7	14	20	35	50
Avg. Time [s]	0.03	<0.01	0.22	0.27	26.21	14.43	176.25	27.30

Another result has to be highlighted. In order to benchmark the SB variants, we additionally used 25 dispatching rules (standard rules including ATC with different parameter settings) to solve each job shop problem instance (note: not the decomposed single machine problems but the whole job shop problem). The best solution obtained by these dispatching rules for a problem instance was denoted as the solution by "Best rule" for this problem

instance. The SB(one pass) version is even worse than the "Best rule". Only when combined with reoptimization (i.e., SB(1, 1)), it became better than the "Best rule". That result confirms the finding in Section 6.3 that reoptimization is a mandatory component for the SB procedure.

Fig. 7 better demonstrates the performance/computational effort ratio of the SB variants. We normalized the results of all SB variants to the result of SB(1,1). It means the normalized 'Opt. Gap' and normalized 'Avg. time' of SB(1,1) equal 1.0. The 66 10x10 problem instances are classified by three different due date tightness factors. In order to examine the effect of due date tightness on SB configurations, we separated the results in three groups corresponding to these three classes. The tighter the due dates, the smaller is the relative improvement of the complex SB variants over SB(1,1). However, that is reasonable, because the harder the problem instance, the less improvement potential exists. Independent of the due date tightness, the relative position of the SB variants to the other are stable. The SB(10,2) dominates the SB(1, VNS) for this class of problem instances. Although the SB(10,3) with its normalized 'Opt. Gap' of 0.59 was better than the SB(VNS,VNS) with a normalized performance of 0.60 for due date tightness of 1.3, the SB(VNS,VNS) was better than SB(10,3) for the other two due date tightness groups. With the regard that the SB(10,3) required more than 6.4 times of computational effort than the SB(VNS,VNS), the SB(VNS,VNS) can dominate the SB(10,3).

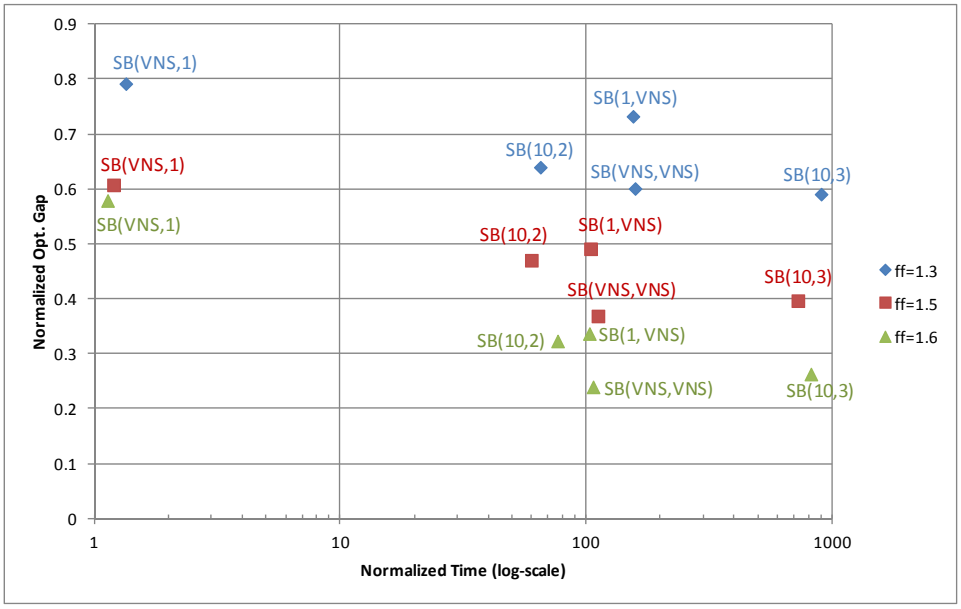


Fig. 7 Performance vs. run time of SB variants for 10x10 problems, normalized to results of SB(1,1)

7 Experiment of large dynamic problem instances

Compared with the previous section, we mainly examine the efficiency of the different components of the SB procedure in this section. For problem instances with 150 jobs and 20 machines, component solutions with high runtime can no longer be considered. For instance, the backtracking method, even the simplest SB(2,1), is incapable for solving one single machine problem with 150 jobs in a reasonable amount of time. Therefore, we focus on the SB(VNS,1), the reoptimization procedure and the VNS for machine planning sequence.

Because the number of jobs of the test instances varies from 100 to 150, the TWT of two instances with different number of jobs are not comparable. Therefore, we use mean weighted tardiness (mean WT) instead of TWT to evaluate the SB variants in this section, which is defined as follows:

$$\text{Mean Weighted Tardiness } (\text{Mean WT}) = \frac{\text{TWT}}{\text{number of jobs of the instance}} \quad (4)$$

7.1 Selection of the dispatching-rule-based SSP

We use the settings of the SB procedure (listed in Table 10) determined by the previous experiment as with high performance and with no or low effects on efficiency. First the scaling parameter k has to be determined. The three rules are applied with various k values. Fig. 8 shows the results. The PINEDORULE has extreme poor performance when k is in the interval [0.001, 2.2]. Fig. 9 shows a detailed view of the low value area of the 'Avg. Gap'. It turns out that the performance of the PINEDORULE does not converge in the interval [0.1, 2.0] like for the 10x10 problem instances. The k value of PINEDORULE strongly depends on the characteristics of the problem instance it is applied on (e.g., the number of jobs and machines, release dates of jobs and machines or the chosen time horizons). This uncertain k setting is a difficulty for applying this rule in practice, when the characteristics of the problem instance are unknown. In contrast, the performance of the other two rules is stable with the k = [0.1, 2.0], like for the 10x10 problem instances. Therefore, these two rules should be preferred to be used for dynamic problem instances. For the further experiments, we use the SUM_ATC_RP rule with $k = 1$.

Table 10 Settings determined by the experiment of static small problem instances

Subproblem solution procedure	PINEDORULE SUM_ATC_RP SUM_ATC_R_2T
MCM	Subproblem Solution
Uncritical machine definition	min TWT

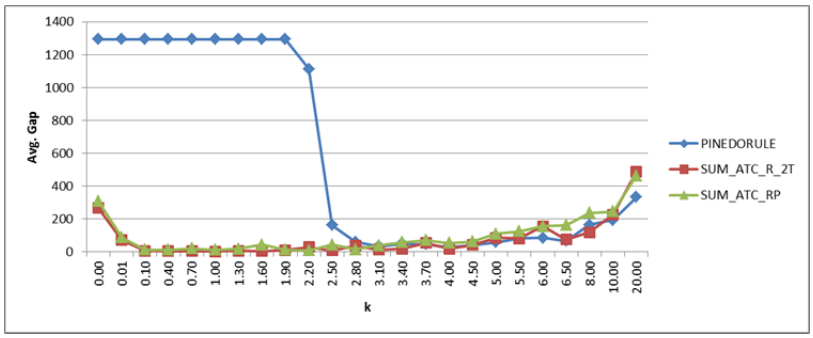


Fig. 8. Overview of the performance of rule-based SSP over different *k*

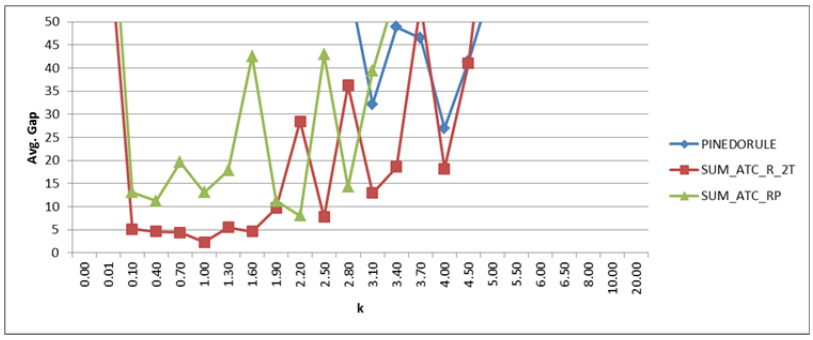


Fig. 9. Fine view of the performance of rule-based SSP over different *k*

7.2 Computational effort of SB components

In order to configure SB variants, the computational effort of each SB component for this class of problem instances have to be firstly identified. We tested the reoptimization component, the VNS as SSP component and the VNS for machine planning component separately. The results are presented in Table 11. Regarding the problem size, we limited the reoptimization cycles to nine for each iteration of the SB procedure. Runs of SB(1,VNS) did not terminate after 8 hours, therefore we limited a run of SB(1,VNS) to 30 minutes. For four of the 40 instances, the SB(1,VNS) terminated within 30 minutes. For the remaining instances, it returned the best solution until it stopped after 30 minutes. As shown in the results, reoptimization is the most effective and efficient component to improve the performance. The performance of SB(1,1) is worse than the 'Best Rule', because the number of reoptimization cycles were limited.

Table 11 Computational effort of SB components for large problem instances

	Reoptimization	Twiddle	Mean WT [min]	Avg. Time [min]
SB(1,1)	Yes (limited to 9)	No	27.60	6.15
SB(VNS,1)	No	No	284.96	2.05
SB(1,VNS)	No	No	216.84	27.59 (limited to 30)
Best Rule	-	-	22.41	<0.1

7.3 Ratio of the performance and the efficiency of SB variants

Regarding the results in Table 11, we configured three SB variants for further evaluations. Table 12 shows the SB variants we configured. First, we combined the both components with the lowest computational effort, namely the SB(VNS,1) and the reoptimization. For the second variant, we used 6 reoptimization cycles and three twiddle cycles, in order to identify, whether twiddle can enhance the reoptimization effectiveness for this class of problems. For the third variant, we apply the VNS for optimizing the machine planning sequence. The time for a run of this variant is limited to 2 hours. The aim of this configuration is to evaluate, whether and how much the VNS for machine planning sequence mechanisms can improve the SB performance. The results are shown in Table 12. All of these variants (i.e. S2, S3, S4) outperformed the Best Rule. Fig. 10 shows the ratio of the performance and computational time of these variants. The values are normalized to the SB(1,1) in Table 11, which is denoted as S1 in Table 12 and Fig. 10. Both twiddle and VNS for machine planning sequence can improve the SB(VNS,1) with reoptimization. However, twiddle can more effectively as well as efficiently enhance the performance of the SB procedure than VNS for machine planning sequence and also dominates the latter. It can first turn out that VNS as SSP should be applied with reoptimization and twiddle for the large instances. More reoptimization cycles expect more performance but more computational time. VNS for machine planning sequence is extreme time consuming, although it is effective to enhance performance. Therefore, it should be only used after no significant performance can be found by reoptimization.

Table 12 Performance and computational time of SB variants

Name	VNS as SSP	Reoptimization	Twiddle	VNS for machine sequence	Mean WT [min]	Avg. Time [min]
S1	No	9	-	-	27.60	6.15
S2	Yes	9	-	-	20.06	9.3
S3	Yes	6	3	-	19.66	12.8
S4	Yes	9	-	2 hours	19.70	74.6

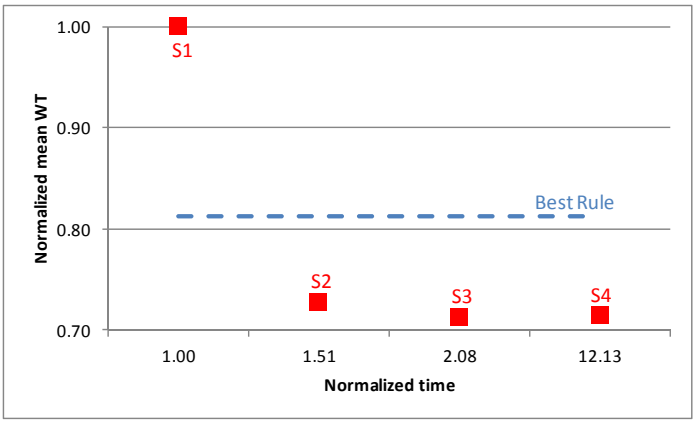


Fig. 10 Ratio of the performance and the computational effort of SB variants for large problems

7.4 Discussion about the efficiency of the SB procedure

The most time consuming SB variant tested in the previous section required a computational effort of about 75 minutes on average. This computational effort seems reasonable, as a high-quality schedule is obtained for the next 24 hours of shop operation (36 hours minus 12 hour overlap with next horizon). If this is not sufficiently fast for a certain application in production scheduling, efficiency of the SB procedure can be improved further by an optimized implementation of the procedure.

Firstly, an implementation in C/C++ or other program languages specific for the hardware of a scheduling system should be more efficient than our Java implementation.

Secondly, the most computational effort of the SB procedure is used for solving the single machine subproblems. Since the SB procedure is a divide-and-conquer approach, individual subproblems can be solved independently at the same time. Therefore, a multithreaded implementation should reduce the computational time of the procedure considerably. For the problem instances with 20 machines, on average 10 subproblems can be solved in parallel. When the hardware supplies 20 processors or threads, the program will finish with only 1/10 effort compared with a single thread implementation.

8 Conclusions and outlook

The shifting bottleneck procedure is a effective heuristic to solve job shop problems, but its various components and parameters severely effect solution quality and runtime and have to be determined. In this paper, we have evaluated the performance as well as efficiency of various components of the shifting bottleneck procedure for the job shop TWT scheduling problem. The evaluation is applied through computational experiments on both static small 10x10 benchmark problem instances and larger dynamic problem instances with 20 machines and about 150 jobs. Our results show that:

a) The PINEDORULE and its two variants SUM_ATC_RP and SUM_ATC_R_2T are the most effective dispatching rules for solving single machine subproblems. However, the scaling parameter k has to be determined. Due to the stable area of their best k-values in [0.1, 2.0] and their time invariance, SUM_ATC_RP and SUM_ATC_R_2T are more suitable for practical applications than the PINEDORULE, especially in dynamic settings.

b) The Variable Neighbourhood Search (VNS) achieves near optimal solutions for subproblems, requires however low computational effort and scales well to larger problem instances. Therefore, it should be considered as the most promising SSP.
c) The simple machine criticality measure 'subproblem solution' outperforms other more complex measures.
d) Reoptimization, potentially together with twiddle, enhances the performance of SB more significantly than a complex subproblem solution procedure and a mechanism for optimizing machine planning sequence. Hence, it has to be considered as mandatory for SB. However, considering its computational effort, the number of reoptimization has to be controlled when solving large problems.
e) The SB(VNS, VNS) using the VNS for machine planning sequence achieves nearly the same performance as the SB backtracking version SB(10,3) for the 10x10 problem instances, but requires a very low computational effort in comparison. It therefore appears to be the most promising configuration for small to medium-sized problem instances.
f) The SB(VNS,1) with reoptimization should be a practical solution for the large problems we tested. However, the number of reoptimizations for each iteration has to be limited.
g) VNS for machine planning sequence no longer terminates in a reasonable time for large problems. It can be used with a certain time limit to improve an initial solution obtained by SB(VNS,1). A new caching scheme is presented to speed-up the use of VNS to optimize the planning sequence.

These are first results to develop guidelines how to tune SB for a given set of problem characteristics using a restricted computational budget. Some conclusions are presented here but more work is currently in progress. In future work, we will investigate more component and parameter combinations of the SB procedure. As next step, we will also examine the SB variants determined to be effective in this paper, e.g., the SB(VNS, VNS), on middle-size problems like 30x10, 50x15. Moreover, since the SB procedure is a divide-and-conquer approach, its single machine subproblems can be solved independently at the same time. Therefore, we will optimize our current implementation of SB procedures to use multithreading and utilize multiple cores of modern processors. This should significantly reduce the computational time of the SB procedure and enable us to examine the SB configurations more intensively and to also investigate more complex SB component solutions on larger problem instances.

Acknowledgements

This research was supported by the International Graduate School for Dynamics in Logistics (IGS) at the University of Bremen

References

1 Pinedo, M., Scheduling: Theory, Algorithms and Systems, Prentice Hall, second ed. (2008)
2 Gordon, V., Strusevich, V. and Dolgui, A., Scheduling with due date assignment under special conditions on job processing, Journal of Scheduling, 15, 447–456 (2012)
3 Singer, M. and Pinedo, M., A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops, IIE Transactions, 30, 109–118 (1998)
4 Pinedo, M. and Singer, M., A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop, Naval Research Logistics (NRL), 46, no. 1, 1–17 (1999)

5 Mason, S. J., Fowler, J. W. and Carlyle, W. M., A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops, Journal of Scheduling, 5, no. 3, 247–262 (2002)
6 Vepsalainen, A. P. J. and Morton, T. E., Priority rules for job shops with weighted tardiness costs, Management Science, 33, no. 8, 1035–1047 (1987)
7 Adams, J., Balas, E. and Zawack, D. The shifting bottleneck procedure for job shop scheduling, Management Science, 34, no. 3, 391–401 (1988)
8 Singer, M., Decomposition methods for large job shops, Computers & Operations Research, 28, no. 3, 193 – 207 (2001)
9 Thompson, G. and Muth, J., Industrial scheduling. Prentice-Hall (1963).
10 Dauzere-Peres, S. and Lasserre, J.-B. A modified shifting bottleneck procedure for job-shop scheduling, International Journal of Production Research, 31, no. 4, 923–932 (1993)
11 Dauzere-Peres, S., A procedure for the one-machine sequencing problem with dependent jobs, European Journal of Operational Research, 81, no. 3, 579 – 589, 1995.
12 Applegate, D. and Cook, W., A computational study of the job-shop scheduling problem, ORSA Journal on Computing, 3, no. 2, 149–156 (1991)
13 Ovacikt, I. M. and Uzsoy, R., A shifting bottleneck algorithm for scheduling semiconductor testing operations, Journal of Electronics Manufacturing, 2, no. 3, 119–134 (1992)
14 Holtsclaw, H. H. and Uzsoy, R., Machine criticality measures and subproblem solution procedures in shifting bottleneck methods: A computational study, The Journal of the Operational Research Society, 47, no. 5, 666–677 (1996)
15 Ovacikt, I. M. and Uzsoy, R., Decomposition Methods for Complex Factory Scheduling Problems, Springer (1997)
16 Demirkol, E., Mehta, S. and Uzsoy, R., A computational study of shifting bottleneck procedures for shop scheduling problems, Journal of Heuristics, 3, 111–137 (1997)
17 Mason, S. J., Fowler, J. W., Carlyle, W. M. and Montgomery, D. C., Heuristics for minimizing total weighted tardiness in complex job shops, International Journal of Production Research, 43, no. 10, 1943–1963 (2005)
18 Scholz-Reiter, B., Hildebrandt, T. and Tan, Y., Effective and efficient scheduling of dynamic job shops - combining the shifting bottleneck procedure with variable neighbourhood search, CIRP Annals - Manufacturing Technology, 62, no. 1, 423–426 (2013)
19 Mönch, L. and Zimmermann, J., Simulation-based assessment of machine criticality measures for a shifting bottleneck scheduling approach in complex manufacturing systems, Computers in Industry, 58, no. 7, 644 – 655 (2007)
20 Bellman, R., On a routing problem, Quart Appl Math, 16, 87–90 (1958)
21 Geiger, M. J., On heuristic search for the single machine total weighted tardiness problem - some theoretical insights and their empirical verification, European Journal of Operational Research, 207, no. 3, 1235 – 1243 (2010)
22 Mladenović, N. and Hansen, P., Variable neighborhood search, Computers & Operations Research, 24, 1097–1100 (1997)
23 Hansen, P. and Mladenović, N., Essays and Surveys in Metaheuristics, ch. Developments of variable neighbourhood search, 415–439. Kluwer Academic Publishers, Boston, Dordrecht, London (2002)
24 Holthaus, O. and Rajendran, C., Efficient jobshop dispatching rules: Further developments, Production Planning & Control, 11, no. 2, 171–178 (2000)
25 Lee, Y. H. and Pinedo, M., Scheduling jobs on parallel machines with sequence-dependent setup times, European Journal of Operational Research, 100, no. 3, 464 – 474 (1997)

MISTA 2013

Local search techniques applied to the Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Time

Lucas U. Rodrigues · Haroldo G. Santos · Túlio A.M. Toffolo · Cristiano L.T.F. Silva

Abstract This work addresses the Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times. In this problem there is a set of jobs and machines, and for each job there is a processing time associated, which is different for each machine. Given two jobs, there is also a setup time that depends on their sequence and on the machine used. The objective considered in this problem is minimizing the makespan. This NP-Hard problem was tackled in this work by local search metaheuristics, such as Simulated Annealing, Iterated Local Search and Late Acceptance Hill Climbing, using several neighborhood structures. These structures are dynamically selected according to their effectiveness in runtime. The parameters of the algorithms were set using the irace package. Computational experiments show that the proposed methods outperform many results found in the literature, improving the best known solutions for several instances.

1 Introduction

This work deals with the unrelated parallel machine scheduling problem with sequence dependent setup time. In this problem, there is a set of jobs and machines, such that each job has an associated time to be executed in each machine. This time may vary depending on the chosen machine. Between two jobs there is also a setup time, which depends on the sequence and also on the machine associated. The objective is minimizing the amount of needed time to execute all jobs.

Let $J = \{j_1, ..., j_n\}$ be the set of n jobs and $M = \{k_1, ..., k_m\}$ be the set of m machines in the problem, such that:

1 : Every job j must be executed only once and by only one machine.

Lucas U. Rodrigues, Haroldo G. Santos and Túlio A.M. Toffolo
Computing Department, Federal University of Ouro Preto, Brazil
E-mail: choa.lucas@gmail.com, haroldo@iceb.ufop.br and tulio@toffolo.com.br

Cristiano L.T.F. Silva
Production Engineer Department, Federal University of Ouro Preto, Brazil
E-mail: cristianoltfs@gmail.com

2 : Each job j has a processing time of p_{kj} if it is executed by the machine k.
3 : There is a time p_{kij} to setup the machine k to execute the job i right after executing the job j.
4 : There is a time p_{kii} to setup the machine k to execute the job i if i is its initial job.

The objective in this problem is to find a solution that allocates all J jobs to the M machines such that the total time to execute all jobs, further called makespan, is the lowest possible.

This problem is usually found in industries and belongs to the $\mathcal{NP}$-hard class (Glass et al, 2001). A complete survey of scheduling problems with setup times is presented by Allahverdi et al (2008). The author presents the main differences and several algorithmic approaches for this class of problems.

The unrelated parallel machine scheduling problem with sequence dependent setup times is also the subject of Rabadi et al (2006), Urlings et al (2010) and Vallada and Ruiz (2011). Rabadi et al (2006) implemented a Greedy Randomized Adaptative Search (Feo and Resende, 1995) style algorithm and also proposed some instances. Later, Vallada and Ruiz (2011) proposed a set of challenging instances for the problem (SOA-2011). The authors implemented a genetic algorithm (Goldberg, 1989) capable of achieving good results within small amount of time.

Follows an example of the problem, P. Table 1 shows the processing times of six jobs in each one of the available machines for the example problem P. The rows represent the machines and the columns represent the jobs.

Table 1 Execution times for the jobs in each machine in the example problem P

Jobs	j_1	j_2	j_3	j_4	j_5	j_6
k_1	2	87	28	32	38	9
k_2	4	21	68	17	43	48

Table 2 shows the setup times for the jobs. In this table, the values of p_{kij} are presented. The rows are the values (jobs) for i and the columns are the values (jobs) for j. The first cell shows the value (machine) for k. The diagonal shows the value for all p_{kii}, which represents the setup for i if this job is the first one executed by the machine. In this example, $p_{kii} = 0 \; \forall \; k \in M, i \in J$.

Table 2 Setup times for example problem P

k_1	j_1	j_2	j_3	j_4	j_5	j_6
j_1	0	1	7	0	3	9
j_2	4	0	7	3	7	8
j_3	7	3	0	2	3	5
j_4	3	8	3	0	5	2
j_5	8	3	7	9	0	5
j_6	8	8	1	2	2	0

k_2	j_1	j_2	j_3	j_4	j_5	j_6
j_1	0	5	1	6	1	7
j_2	6	0	7	7	6	2
j_3	7	6	0	9	6	9
j_4	3	7	3	0	1	7
j_5	5	8	5	6	0	9
j_6	7	4	1	7	9	0

Figure 1 shows a greedy solution for the example problem P. In this solution, the jobs 1, 6, 3 and 5 are executed, in this order, by the machine M_1, while the jobs 4 and 2 are executed by machine M_2. The white boxes represent the processing time of a job,

while the gray boxes represent the setup times between two jobs. Figure 2 shows the optimal solution for this problem. When compared to the solution shown by Figure 1, the optimal solution reduced the makespan from 89 to 74 units of time, improving the solution by 15 units of time.

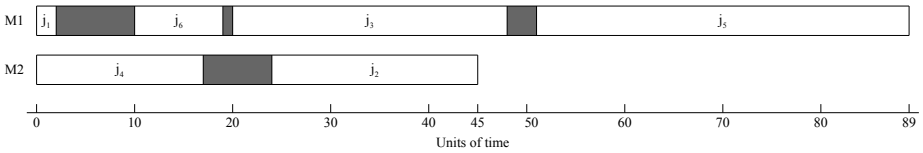


Fig. 1 Sample greedy solution for the problem P

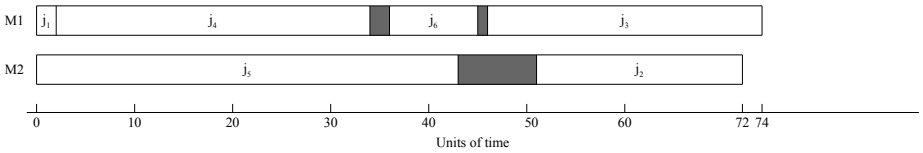


Fig. 2 Optimal solution for the problem P

2 Greedy constructive method

In order to obtain an initial feasible solution, a simple greedy algorithm was developed. The algorithm consists in the allocation of each job to the best machine for the current situation. In the beginning, a list with all the unallocated jobs is created. Then, each job in this list is allocated, one by one, at the end of the queue of the machine that incurs in the smallest increase to the current makespan. The procedure stops when all the jobs are allocated.

3 Neighborhoods Structures

To explore the search space for different solutions, five neighborhood structures were developed. These neighborhoods are used in the subroutines of the implemented metaheuristics during the local search phase.

The neighborhood to be used in each step of the algorithms is chosen in a probabilist way. The probability of selecting one of the q neighborhoods is $p_i,\ i \in \{1, ..., q\}$. Initially, all priorities receive the same value, i.e. $p_i = 1/q\ \forall i \in \{1, ..., q\}$, so that all the neighborhoods have even chances of being selected. Our methods update these probabilities in an adaptive way considering the history of the search process.

The update procedure of the priorities is executed every $Itmax$ iterations, and it consists in giving higher priorities to the most promising neighborhoods. For that, the

sum of improvements incurred by each neighborhood is stored in a variable α_i, $i \in \{1, ..., q\}$. The improvement is computed, whenever the incumbent solution is updated, as the difference between the cost of the best solution, $f(s^*)$, and the cost of the solution generated, $f(s')$, i.e. $\Delta = f(s^*) - f(s')$. When the update procedure is called, the priorities are updated if and only if at least one of the iterations improved the best solution. If that is the case, then $p_i = \alpha_i / \sum_{j \in \{1,...,q\}} \alpha_j \ \forall i \in \{1, ..., q\}$ and the variables α are reseted, i.e. $\alpha_i = 0 \ \forall i \in \{1, ..., q\}$.

The following subsections detail the developed neighborhoods structures. It is important to note that we used the irace package (Birattari et al, 2007; Balaprakash et al, 2007) to determine whether it is interesting to (1) allow moves to use any machines or (2) restrict the moves to include the machine with the largest total execution time. For most instances, the first option is preferable for three of the neighborhoods while the second option is preferable for the other two neighborhoods.

3.1 Task Move Neighborhood

A neighbor in the Task Move Neighborhood is generated by moving one job from one machine, M_x, to another machine, M_y. Figure 3 shows an example of the generation of a Task Move neighbor.

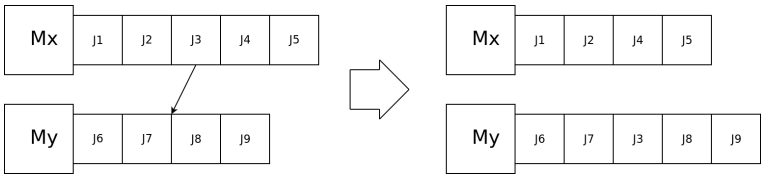


Fig. 3 Example of a Neighbor in the Task Move Neighborhood

3.2 Shift Neighborhood

A neighbor in the Shift Neighborhood is generated by re-scheduling one job from the machine with the largest total execution time, M_x, to another position in the machine. Figure 4 shows an example of how a Task Move neighbor is generated.

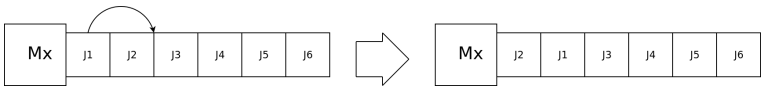


Fig. 4 Example of a Neighbor in the Task Shift Neighborhood

3.3 Switch Neighborhood

A neighbor in the Switch Neighborhood structure is generated by switching the order of two jobs in any machine, M_x. Figure 5 shows an example of a neighbor in this structure.

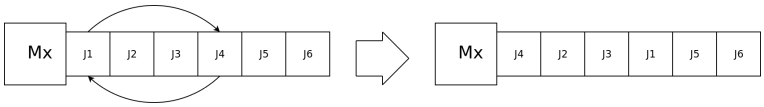


Fig. 5 Example of a Neighbor in the Switch Neighborhood

3.4 Swap Neighborhood

A neighbor in the Swap Neighborhood is generated by swapping two jobs between two machines, M_x and M_y. Figure 6 presents an example of a neighbor in this structure. It is important to note that the swapped jobs may be placed in any position on the other machine.

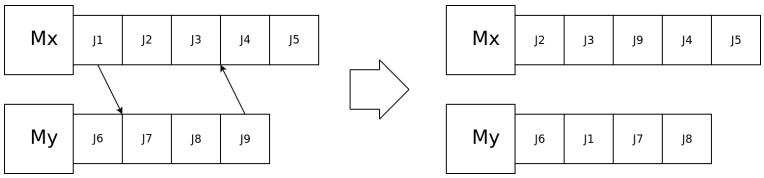


Fig. 6 Example of a Neighbor in the Swap Neighborhood

3.5 2-Realloc Neighborhood

This neighborhood structure considers, again, the machine with the largest total execution time, M_x. To generate a neighbor, two jobs being executed by the machine M_x have their order switched to random positions. An example is shown in Figure 7.

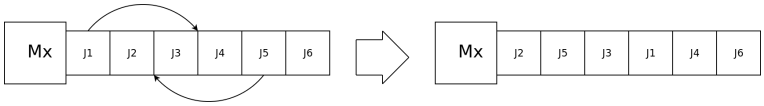


Fig. 7 Example of a Neighbor in the 2-Realloc Neighborhood

4 Heuristics Methods

Three metaheuristics were implemented to tackle the problem. The first one is the Simulated Annealing (SA) method, the second one is the Iterated Local Search (ILS) algorithm and the third one is a Late Acceptance Hill Climbing (LAHC) procedure. The three methods use the neighborhood structures presented in Section 3. The following sections detail the implemented metaheuristics.

4.1 Simulated Annealing

Proposed by Kirkpatrick et al (1983), the metaheuristic Simulated Annealing is a probabilistic method based on an analogy to thermodynamics simulating the cooling of a set of heated atoms. This technique starts its search from any initial solution. The main procedure consists of a loop that randomly generates, at each iteration, one neighbor s' of the current solution s. Let Δ be the variation of the objective function value incurred from moving to the candidate neighbor ($\Delta = f(s') - f(s)$). The method accepts the move and the neighbor becomes the new current solution if $\Delta < 0$. On the other hand, if $\Delta \geq 0$ then the neighbor may be accepted with a probability $e^{-\Delta/T}$, where T is a method parameter, called temperature and regulates the probability of accepting worse solutions.

The temperature may assume, initially, a high value T_0. After a fixed number of iterations, SA_{max} (that represents the number of iterations needed to the system reaches the thermic equilibrium at a given temperature), the temperature is gradually lowered by a cooling rate α, such that $T_k \leftarrow \alpha \times T_{k-1}$, and $0 < \alpha < 1$. With this procedure, a greater chance to avoid local minimum occurs at the initial iteration and as T approaches zero, the algorithm behave like a descent method as it reduces the likelihood of accepting worsening moves (Glover and Kochenberger, 2003).

Algorithm 1 shows the pseudo-code of the Simulated Annealing implemented. In this algorithm, $f(.)$ is the objective function, $N_k(.)$ is the neighborhood structure k, and the method $selectNeighborhood(.)$ returns one of the neighborhoods described in section 3, according to their probability (dynamically calculated). The algorithm has the following parameters:

- s: an initial solution
- T_0: initial temperature
- α: cooling rate
- SA_{max}: number of iteration in each temperature
- $timeout$: time limit

One of the main issues of the Simulated Annealing is its high number of parameters. The first one, T_0, is critical but can be estimated automatically. Ben-Ameur (2004) points some ways to calculate the initial temperature to obtain a given acceptance ratio. In this work, we use a simple method to calculate the value for T_0. Given an initial solution, s, a number dn of different neighbors are analyzed and the worst Δ obtained is stored. This Δ is basically a value for T_0 in which all of the dn neighbors analyzed would be accepted. In this work, a ratio of 90% of acceptance is considered, and so the worst Δ is multiplied by 0.9. The pseudo-code of the implemented method is shown in Algorithm 2. This algorithm has three parameters: (i) the initial solution s, (ii) the number dn of neighbors to be generated and (iii) the $ratio$ of acceptance. In this work, all tests ran with $dn = 100$ and $ratio = 0.9$.

Algorithm 1: Simulated Annealing

```
Input: s, T_0, α, SAmax, timeout
Output: Best solution s* found.
s* ← s;
T ← T_0;
IterT ← 0; while elapsedTime < timeout do
    while IterT < SAmax do
        IterT ← IterT + 1;
        k ← selectNeighborhood();
        Generate a random neighbor s' ∈ N_k(s);
        Δ = f(s') − f(s);
        if Δ < 0 then
            s ← s';
            if f(s') < f(s*) then s* ← s';
        else
            Take x ∈ [0, 1];
            if x < e^{−Δ/T} then s ← s';
    T ← α × T;
    IterT ← 0;
    if T < 0.01 then
        T ← T_0
return s*;
```

Algorithm 2: Calculating T_0 for Simulated Annealing

```
Input: s, dn, ratio
Output: Initial time, T_0, for Simulated Annealing
i ← 0;
Δ ← 0;
for i ← 0 to dn do
    k ← selectNeighborhood();
    Generate a random neighbor s' ∈ N_k(s);
    Δ_tmp = f(s') − f(s);
    if Δ_tmp > Δ then
        Δ ← Δ_tmp
return Δ * ratio;
```

4.2 Iterated Local Search

The method Iterated Local Search (ILS) (Lourenco et al, 2003) is based on the idea that a local search procedure can achieve better results by optimizing different solutions generated through disturbances on the local optimum solution.

The ILS algorithm starts from an initial solution s_0 and makes disturbances of size p_{size} under s_0, followed by a descent method. A disturbance is the unconditional acceptance of a neighbor generated by any one of the neighborhoods presented in section 3.

The descent phase uses a Random Non-Ascendant Method, which accepts neighbors only if their objective function value is better or equal than the current one. Tested moves are excluded from the neighborhood and return only when an improvement on the current best solution is reached. The local search phase ends when there is no remaining neighbor to be explored.

The local search produces a solution s' which will be accepted if it is better than the best solution s^* found. In such case, the disturbance size p_{size} gets back to the initial size p_0. If the iteration $Iter$ reaches a limit $Iter_{max}$, the disturbance size is incremented. Yet, if the disturbance size reaches a bound p_{max}, it goes back to the initial size p_0.

Algorithm 3 presents the pseudo-code of the implemented ILS. In this algorithm, $f(.)$ is the objective function, $N_k(.)$ the neighborhood structure k, $selectNeighborhood(.)$ a method that returns one of the neighborhoods described in section 3 at random and $descentPhase$ represents the descent phase. The descent phase is a search for improvements in all the neighborhoods until a local optima is found. In this search the neighborhoods are selected sequentially and the neighbors are enumerated. The ILS algorithm has the following parameters:

- s: initial feasible solution
- p_0: number of movements made in the initial perturbation level
- p_{max}: maximum number of movements made in one perturbation
- ILS_{max}: number of iterations
- $timeout$: time limit

Algorithm 3: Iterated Local Search

```
Input: s, ILS_max, p_0, p_max, timeout
Output: Best solution s* found.
s ← descentPhase(s);
s* ← s;
p_size ← p_0;
iter ← 0;
while elapsedTime < timeout do
    for j ← 0 to p_size do
        k ← selectNeighborhood();
        Generate a random neighbor s' ∈ N_k(s);
        s ← s';
    s' ← descentPhase(s);
    if f(s') < f(s*)) then
        s ← s';
        s* ← s';
        Iter ← 0;
        p_size ← p_0;
    else
        s ← s*;
        Iter ← Iter + 1;
    if Iter ≥ ILS_max then
        p_size ← p_size + p_0;
        if p_size ≥ p_max then p_size ← p_0;
return s*;
```

4.3 Late Acceptance Hill-Climbing

The Late Acceptance Hill-Climbing (LAHC) metaheuristic was recently proposed by Burke and Bykov (2008). It consists in an adaption of the classical Hill-Climbing

method. It relies on comparing a new candidate solution with the last l solutions considered in the past before accepting or rejecting it. Note that the candidate solution may be accepted even if it is worse than the current solution, since it is compared to the current solution at the i-th (past) iteration.

This metaheuristic was created with three goals in mind: to be a one-point search procedure that does not employ an artificial cooling schedule like Simulated Annealing does; to effectively use the information collected during previous iterations of the search; and to employ a simple acceptance mechanism (i.e. almost as simple as Hill-Climbing) (Burke and Bykov, 2012).

In this method, we store a list $f'_k = \{f_0, ..., f_{l-1}\}$ of costs of solutions. Initially this list is filled with the cost of the initial solution s: $f'_k \leftarrow f(s)\ \forall k \in \{0, ..., l-1\}$. At each iteration i, a candidate solution s' is generated. The candidate solution is accepted if its cost is smaller or equals to the cost stored at i mod l position of the list f'. Moreover, if this solution is better than the best solution found, it is updated: $s^* \leftarrow s'$. Afterwards, the position $v = i$ mod l of f' is updated: $f'_v \leftarrow f(s')$. This process repeats until a stop condition is reached.

Algorithm 4 presents our implementation of the LAHC.

Algorithm 4: Late Acceptance Hill Climbing

```
Input: s, l
Output: Best solution s* found.
f'_k ← f(s) ∀k ∈ {0, ..., l - 1};
s* ← s;
i ← 0;
while elapsedTime < timeout do
    k ← selectNeighborhood();
    Generate a random neighbor s' ∈ N_k(s);
    v ← i mod l;
    if f(s') ≤ f'_v then
        s ← s';
        if f(s) < f(s*) then
            s* ← s;
    f'_v ← f(s);
    i ← i + 1;
return s*;
```

5 Experiments

In this section the computational experiments are detailed. The instances from Vallada and Ruiz (2011) were used to evaluate the implemented algorithms. Vallada and Ruiz (2011) proposed a total of 1,640 different instances. During the tests, though, it was observed that most of the instances with the same size had similar behavior. Because of that, in this work we report the results for only 90 of these instances, considering the ones with smallest range (1 to 9) and largest range (1 to 124) for values of execution and setup times. This set of 90 instances has several different values for number of jobs, $n = \{6, 8, 10, 12, 50, 100, 150, 200, 250\}$, and number of machines,

$m = \{2, 3, 4, 5, 10, 15, 20, 25, 30\}$. The solutions generated for all the 1,640 instances, as well as reports and the software developed, are available in the GOAL-UFOP website[1].

The algorithms were coded in C++ and compiled with G++ version 4.6.3. The experiments ran on a computer with an Intel Core i5 2.0Ghz processor and 8Gb of RAM memory, running Linux Ubuntu 12.04.

The instances are named according to the following pattern: I-n-m-S-a-b-1. In this pattern, I and S are irrelevant in this context and may be ignored, n is the number of jobs, m is the number of machines and, finally, a and b represent the range $[a, b]$ of possible values for the execution and setup times in the instance, respectively.

To calibrate the parameters of the algorithms, the irace package was used. This package implements the iterated racing procedure, which is an extension of Iterated F-race (I/F-Race), proposed by Birattari et al (2007) and further developed by Yuan et al (2010). The main use of irace is the automatic configuration of optimization algorithms, that is, finding the most appropriate settings of an optimization. Irace is implemented as an R package (R Development Core Team, 2008) and it builds upon the race package by Birattari (2003).

After several tests with irace, the chosen parameters were:

- Simulated Annealing (SA): $SA_{max} \leftarrow 2850$ and $\alpha \leftarrow 0.95$.
- Iterated Local Search (ILS): $ILS_{max} \leftarrow 5245$ and $p_0 \leftarrow 1$.
- Late Acceptance Hill Climbing (LAHC): $l \leftarrow 100$.

Table 3 presents the best results obtained by the algorithms after 10 minutes of runtime. This table shows that 45 instances from the literature had their best known solutions improved. The algorithms LAHC and ILS obtained the best results, while the algorithm SA obtained poor results for the largest instances.

Table 4 shows the best solution, the average and the standard deviation of 10 runs of the algorithms with 1 minute of runtime.

6 Conclusions and Future Work

This work presented three heuristic approaches for the Unrelated Parallel Machine Scheduling Problem with Sequence Dependent Setup Times. The proposed algorithms uses five different neighborhoods, which are dynamically chosen during the execution. These neighborhoods, as well as the parameters of the algorithms, were tunned using the irace package.

Several benchmark instances by Vallada and Ruiz (2011) were considered. The proposed methods were able to improve the best known solution for 45 instances (out of the 90 considered) and found at least the best known solution for the other 45 instances. The LAHC and ILS algorithms produced the best results for these instances. In general, ILS outperformed LAHC when dealing with larger instances, while the opposite happened with most smaller instances.

For future work, we suggest a careful analysis of the tuning results obtained with irace. In this work, we considered the same parameters for all the instances, small and large ones. But it is expected that the best parameters for small instances will not be the best for large instances. It may also be interesting to develop other neighborhoods to do a better exploration of the search space, as well as to implement and test other metaheuristics.

[1] Code, solutions and instances: `http://www.goal.ufop.br/softwares/jobshop`

Table 3 Best results after 10 minutes of runtime

	Results					
Instances	SA	ILS	LAHC	Our Best	Best SOA	Improved?
I-6-2-S-1-124_1	164	164	164	164	164	
I-6-2-S-1-9_1	74	74	74	74	74	
I-6-3-S-1-124_1	119	119	119	119	119	
I-6-3-S-1-9_1	52	52	52	52	52	
I-6-4-S-1-124_1	77	77	77	77	77	
I-6-4-S-1-9_1	42	42	42	42	42	
I-6-5-S-1-124_1	52	52	52	52	52	
I-6-5-S-1-9_1	52	52	52	52	52	
I-8-2-S-1-124_1	211	211	211	211	211	
I-8-2-S-1-9_1	88	88	88	88	88	
I-8-3-S-1-124_1	140	140	140	140	140	
I-8-3-S-1-9_1	72	72	72	72	72	
I-8-4-S-1-124_1	106	106	106	106	106	
I-8-4-S-1-9_1	62	62	62	62	62	
I-8-5-S-1-124_1	55	55	55	55	55	
I-8-5-S-1-9_1	43	43	43	43	43	
I-10-2-S-1-124_1	288	288	293	288	288	
I-10-2-S-1-9_1	169	170	170	169	169	
I-10-3-S-1-124_1	136	136	136	136	136	
I-10-3-S-1-9_1	79	79	79	79	79	
I-10-4-S-1-124_1	111	111	111	111	111	
I-10-4-S-1-9_1	67	67	67	67	67	
I-10-5-S-1-124_1	91	91	91	91	91	
I-10-5-S-1-9_1	91	91	91	91	91	
I-12-2-S-1-124_1	202	202	225	202	202	
I-12-2-S-1-9_1	264	264	264	264	264	
I-12-3-S-1-124_1	239	239	242	239	239	
I-12-3-S-1-9_1	140	140	140	140	140	
I-12-4-S-1-124_1	111	111	111	111	111	
I-12-4-S-1-9_1	79	79	79	79	79	
I-12-5-S-1-124_1	91	91	91	91	91	
I-12-5-S-1-9_1	80	80	80	80	80	
I-50-10-S-1-124_1	121	122	114	114	114	
I-50-10-S-1-9_1	69	70	67	67	67	
I-50-15-S-1-124_1	77	75	75	75	75	
I-50-15-S-1-9_1	34	34	34	34	36	yes
I-50-20-S-1-124_1	56	52	46	46	52	yes
I-50-20-S-1-9_1	30	33	28	28	31	yes
I-50-25-S-1-124_1	33	33	30	30	37	yes
I-50-25-S-1-9_1	22	22	22	22	22	
I-50-30-S-1-124_1	29	29	21	21	27	yes
I-50-30-S-1-9_1	11	11	11	11	11	
I-100-10-S-1-124_1	273	224	212	212	221	yes
I-100-10-S-1-9_1	117	117	117	117	131	yes
I-100-15-S-1-124_1	174	137	128	128	141	yes
I-100-15-S-1-9_1	68	65	63	63	68	yes
I-100-20-S-1-124_1	129	96	88	88	100	yes
I-100-20-S-1-9_1	46	43	42	42	46	yes
I-100-25-S-1-124_1	99	74	66	66	73	yes
I-100-25-S-1-9_1	30	28	28	28	31	yes
I-100-30-S-1-124_1	76	53	45	45	61	yes
I-100-30-S-1-9_1	29	27	27	27	29	yes
I-150-10-S-1-124_1	408	323	323	323	332	yes
I-150-10-S-1-9_1	188	186	183	183	184	yes
I-150-15-S-1-124_1	258	194	190	190	210	yes
I-150-15-S-1-9_1	98	91	91	91	97	yes
I-150-20-S-1-124_1	184	130	124	124	141	yes
I-150-20-S-1-9_1	76	67	69	67	75	yes
I-150-25-S-1-124_1	145	106	95	95	112	yes
I-150-25-S-1-9_1	48	43	45	43	48	yes
I-150-30-S-1-124_1	117	83	71	71	85	yes
I-150-30-S-1-9_1	37	32	36	32	36	yes
I-200-10-S-1-124_1	512	393	400	393	417	yes
I-200-10-S-1-9_1	266	250	253	250	260	yes
I-200-15-S-1-124_1	316	234	223	223	253	yes
I-200-15-S-1-9_1	136	125	127	125	131	yes
I-200-20-S-1-124_1	234	170	165	165	176	yes
I-200-20-S-1-9_1	83	73	77	73	78	yes
I-200-25-S-1-124_1	182	125	127	125	141	yes
I-200-25-S-1-9_1	63	55	61	55	61	yes
I-200-30-S-1-124_1	140	102	106	102	122	yes
I-200-30-S-1-9_1	51	42	50	42	49	yes
I-250-10-S-1-124_1	605	483	498	483	500	yes
I-250-10-S-1-9_1	293	271	282	271	278	yes
I-250-15-S-1-124_1	350	279	292	279	313	yes
I-250-15-S-1-9_1	157	145	147	145	150	yes
I-250-20-S-1-124_1	275	219	218	218	232	yes
I-250-20-S-1-9_1	106	94	101	94	100	yes
I-250-25-S-1-124_1	213	155	162	155	175	yes
I-250-25-S-1-9_1	78	69	75	69	76	yes
I-250-30-S-1-124_1	158	127	133	127	138	yes
I-250-30-S-1-9_1	62	50	61	50	57	yes

Table 4 Results after 10 runs with 1 minute of runtime

	SA			ILS			LAHC		
Instances	best	avg	st.dev.	best	avg	st.dev.	best	avg	st.dev.
I-6-2-S-1-124_1	164	164	0.00	164	164	0.00	164	166	3.90
I-6-2-S-1-9_1	74	74	0.00	74	74	0.00	74	74	0.00
I-6-3-S-1-124_1	119	119	0.00	119	119	0.00	119	120	0.50
I-6-3-S-1-9_1	52	52	0.00	52	52	0.00	52	52	0.00
I-6-4-S-1-124_1	77	77	0.00	77	77	0.00	77	77	0.00
I-6-4-S-1-9_1	42	42	0.00	46	46	0.00	42	44	2.00
I-6-5-S-1-124_1	52	52	0.00	52	52	0.00	52	52	0.00
I-6-5-S-1-9_1	52	52	0.00	52	52	0.00	52	52	0.00
I-8-2-S-1-124_1	211	211	0.00	211	212	1.64	211	212	1.22
I-8-2-S-1-9_1	88	88	0.00	88	88	0.00	88	89	1.30
I-8-3-S-1-124_1	140	140	0.00	140	141	2.68	140	140	0.00
I-8-3-S-1-9_1	72	72	0.00	72	72	0.00	72	72	0.00
I-8-4-S-1-124_1	106	106	0.00	106	106	0.00	106	108	2.60
I-8-4-S-1-9_1	62	62	0.00	62	62	0.00	62	65	4.33
I-8-5-S-1-124_1	55	55	0.00	55	60	12.52	55	60	7.79
I-8-5-S-1-9_1	43	43	0.00	43	43	0.00	43	43	0.00
I-10-2-S-1-124_1	288	288	0.00	288	290	4.47	293	300	8.38
I-10-2-S-1-9_1	169	169	0.00	169	169	0.45	169	169	0.00
I-10-3-S-1-124_1	136	136	0.00	136	140	4.02	136	139	3.00
I-10-3-S-1-9_1	79	79	0.00	79	79	1.79	79	79	0.43
I-10-4-S-1-124_1	111	111	0.00	111	114	7.16	111	113	1.79
I-10-4-S-1-9_1	67	67	0.00	67	67	0.00	67	67	0.00
I-10-5-S-1-124_1	91	91	0.00	91	96	9.55	91	97	8.67
I-10-5-S-1-9_1	91	91	0.00	91	91	0.00	91	91	0.00
I-12-2-S-1-124_1	202	202	0.00	202	202	0.00	225	235	8.95
I-12-2-S-1-9_1	264	264	0.00	264	264	0.00	264	267	2.69
I-12-3-S-1-124_1	239	239	0.00	239	245	10.50	242	245	1.92
I-12-3-S-1-9_1	140	140	0.00	140	140	0.00	140	142	1.22
I-12-4-S-1-124_1	111	111	0.00	111	115	7.38	111	123	12.03
I-12-4-S-1-9_1	79	79	0.00	79	79	0.89	79	84	5.02
I-12-5-S-1-124_1	91	91	0.00	91	96	5.50	91	99	5.68
I-12-5-S-1-9_1	80	80	0.00	80	81	1.92	80	80	0.00
I-50-10-S-1-124_1	133	135	3.29	121	126	6.18	114	116	1.48
I-50-10-S-1-9_1	69	71	1.67	70	72	2.41	67	68	0.71
I-50-15-S-1-124_1	93	95	2.45	77	84	5.59	75	78	2.74
I-50-15-S-1-9_1	35	35	0.55	34	35	1.14	34	35	0.50
I-50-20-S-1-124_1	62	65	2.41	56	57	1.52	46	50	2.96
I-50-20-S-1-9_1	30	30	0.45	33	33	0.00	28	29	0.43
I-50-25-S-1-124_1	42	43	1.22	33	36	2.59	30	31	0.87
I-50-25-S-1-9_1	22	22	0.00	22	22	0.00	22	22	0.00
I-50-30-S-1-124_1	33	33	0.45	29	30	1.14	21	21	0.00
I-50-30-S-1-9_1	11	11	0.55	11	11	0.55	11	11	0.00
I-100-10-S-1-124_1	280	287	6.91	237	242	4.83	212	223	8.41
I-100-10-S-1-9_1	121	124	2.00	117	118	1.52	117	118	0.83
I-100-15-S-1-124_1	183	186	3.11	143	146	2.79	128	134	5.36
I-100-15-S-1-9_1	68	69	1.10	66	66	1.10	63	64	0.50
I-100-20-S-1-124_1	129	130	1.95	96	110	9.86	88	90	1.41
I-100-20-S-1-9_1	46	47	0.89	43	44	1.00	42	42	0.43
I-100-25-S-1-124_1	99	101	1.95	74	77	2.95	66	67	0.83
I-100-25-S-1-9_1	31	31	0.00	28	29	1.30	28	29	0.50
I-100-30-S-1-124_1	76	79	2.88	56	61	3.27	45	49	3.27
I-100-30-S-1-9_1	29	29	0.55	27	27	0.45	27	27	0.43
I-150-10-S-1-124_1	408	417	5.87	336	342	4.00	323	328	4.56
I-150-10-S-1-9_1	191	194	2.24	186	188	1.79	183	185	1.92
I-150-15-S-1-124_1	258	258	0.00	194	206	7.92	190	195	2.87
I-150-15-S-1-9_1	99	100	1.22	91	92	1.79	91	92	0.83
I-150-20-S-1-124_1	189	194	3.91	130	139	6.89	124	127	3.27
I-150-20-S-1-9_1	76	79	2.12	68	70	2.30	69	70	0.43
I-150-25-S-1-124_1	149	152	3.83	108	110	3.27	95	96	0.83
I-150-25-S-1-9_1	49	50	1.00	43	44	0.89	45	46	0.83
I-150-30-S-1-124_1	120	120	0.00	84	87	2.07	71	73	1.41
I-150-30-S-1-9_1	37	38	1.52	34	34	0.45	36	37	0.71
I-200-10-S-1-124_1	512	512	0.00	402	416	9.89	400	418	11.82
I-200-10-S-1-9_1	269	271	1.30	251	252	1.14	253	255	1.92
I-200-15-S-1-124_1	316	331	8.50	241	247	8.57	223	231	5.93
I-200-15-S-1-9_1	136	138	1.87	125	126	1.30	127	128	0.71
I-200-20-S-1-124_1	234	234	0.00	172	179	6.73	165	169	3.63
I-200-20-S-1-9_1	83	85	1.41	74	76	1.82	77	78	0.43
I-200-25-S-1-124_1	182	182	0.00	136	140	3.21	127	129	3.90
I-200-25-S-1-9_1	63	65	1.95	56	56	1.30	61	61	0.43
I-200-30-S-1-124_1	140	148	4.47	104	108	3.35	106	107	1.00
I-200-30-S-1-9_1	51	52	0.89	44	44	0.55	50	51	0.50
I-250-10-S-1-124_1	605	605	0.00	490	498	7.96	498	504	6.73
I-250-10-S-1-9_1	299	300	2.51	274	279	3.58	282	285	3.42
I-250-15-S-1-124_1	350	350	0.00	284	296	7.78	292	299	7.18
I-250-15-S-1-9_1	159	161	2.79	145	146	1.30	147	149	1.58
I-250-20-S-1-124_1	275	291	9.39	220	224	3.81	218	222	3.04
I-250-20-S-1-9_1	106	107	1.52	94	95	1.48	101	101	0.43
I-250-25-S-1-124_1	213	213	0.00	160	162	1.79	162	165	1.66
I-250-25-S-1-9_1	80	83	2.39	69	70	1.10	75	77	1.12
I-250-30-S-1-124_1	158	158	0.00	128	130	1.79	133	135	1.48
I-250-30-S-1-9_1	62	63	0.71	50	51	1.34	61	62	0.50

Acknowledgements We acknowledge FAPEMIG and CNPq for supporting the development of this research. We would like to also thank the anonymous revisers of this paper for their detailed and very helpful comments.

References

Allahverdi A, Ng C, Cheng T, Kovalyov MY (2008) A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187(3):985–1032, DOI 10.1016/j.ejor.2006.06.060, URL http://linkinghub.elsevier.com/retrieve/pii/S0377221706008174

Balaprakash P, Birattari M, Sttzle T (2007) Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In: Hybrid Metaheuristics, 4th International Workshop, HM 2007, LNCS, vol 4771, Springer-Verlag, Berlin, Germany, pp 108–122

Ben-Ameur W (2004) Computing the Initial Temperature of Simulated Annealing. Computational Optimization and Applications 29:369–385, DOI 10.1023/B:COAP.0000044187.23143.bd

Birattari M (2003) The race package for R. Racing methods for the selection of the best. Tech. Rep. TR/IRIDIA/2003-037, IRIDIA, Universit Libre de Bruxelles, Brussels, Belgium

Birattari M, Balaprakash P, Dorigo M (2007) The ACO/F-race algorithm for combinatorial optimization under uncertainty. In: Metaheuristics–Progress in Complex Systems Optimization, Operations Research/Computer Science Interfaces Series, Springer, Berlin, Germany, pp 189–203

Burke EK, Bykov Y (2008) A Late Acceptance Strategy in Hill-Climbing for Exam Timetabling Problems. In: PATAT '08 Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, URL http://w1.cirrelt.ca/~{}patat2008/PATAT_7_PROCEEDINGS/Papers/Bykov-HC2a.pdf

Burke EK, Bykov Y (2012) The late acceptance hill-climbing heuristic. Tech. Rep. CSM-192, Department of Computing Science and Mathematics, University of Stirling

Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. In: Journal of Global Optimization, pp 6, 109–133

Glass C, Potts C, Strusevich V (2001) Scheduling batches with sequential job processing for two-machine flow and open shops. INFORMS J on Computing 13(2):120–137, DOI 10.1287/ijoc.13.2.120.10521, URL http://dx.doi.org/10.1287/ijoc.13.2.120.10521

Glover F, Kochenberger G (2003) Handbook of Metaheuristics. International Series in Operations Research & Management Science, Kluwer Academic Publishers, URL http://books.google.com.br/books?id=0_10T_KeqOgC

Goldberg DE (1989) Genetic Algorithms in Search. Optimization and Machine Learning. Addison-Wesley. Berkeley

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680

Lourenco HR, Martin OC, Stutzle T (2003) Iterated local search. In: Glover F, Kochenberger G (eds) Handbook of Metaheuristics, Kluwer Academic Publishers, Boston, chap 11

R Development Core Team (2008) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL http://www.R-project.org, ISBN 3-900051-07-0

Rabadi G, Moraga R, Al-Salem A (2006) Heuristics for the unrelated parallel machine scheduling problem with setup times. Journal of Intelligent Manufacturing 17:85–97, DOI 10.1007/s10845-005-5514-0, URL http://dx.doi.org/10.1007/s10845-005-5514-0

Urlings T, Ruiz R, Serifoglu FS (2010) Genetic algorithms with different representation schemes for complex hybrid flexible flow line problems. Int J Metaheuristics 1(1):30–54, DOI 10.1504/IJMHEUR.2010.033122, URL http://dx.doi.org/10.1504/IJMHEUR.2010.033122

Vallada E, Ruiz R (2011) A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. European Journal of Operational Research 211(3):612 – 622, DOI 10.1016/j.ejor.2011.01.011, URL http://www.sciencedirect.com/science/article/pii/S0377221711000142

Yuan Z, Sttzle T, Birattari M (2010) MADS/F-Race: Mesh adaptive direct search meets F-Race. In: Garca-Pedrajas N, Francisco H, Fyfe C, Bentez JM, Ali M (eds) Trends in Applied Intelligent Systems, 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2010, LNAI, vol 6096, Springer-Verlag, Berlin, Germany, pp 41–50

MISTA 2013

An Iterative Local Search Algorithm for Scheduling Precedence-Constrained Applications on Heterogeneous Machines

Aurelio A. Santiago Pineda · Johnatan E. Pecero · Héctor J. Fraire Huacuja · Juan J. Gonzalez Barbosa · Pascal Bouvry

Abstract The paper deals with the problem of scheduling precedence-constrained applications on a distributed heterogeneous computing system with the aim of minimizing the response time or total execution time. The main contribution is a scheduling algorithm that promotes an iterative local search process. Due to a lack of generally accepted standard benchmarks for the evaluation of scheduling algorithms in the heterogeneous computing systems we also generate a benchmark of synthetic instances. The benchmark is composed of small size synthetic deterministic non-preemptive program graphs known in the literature. We compute the optimal solution and the global optimal value with an exact enumerative search method that explores all possible solutions. We compare the performance of the proposed local search algorithm with the optimal values. We have simulated the proposed algorithm using graphs obtained from real-world applications emphasizing the interest of the approach.

1 Introduction

Heterogeneous computing systems are a commonplace infrastructure that provides resources in a distributed way interconnected via networks for executing parallel applications with a large amount of data and computing power. In such a system, a parallel application can be partitioned into a number of cooperative tasks that are distributed to the resources for parallel execution. However, the performance of a parallel application executed on a parallel and distributed computing system heavily depends on the scheduling of the tasks from the application onto the available processors in the system, which if not properly solved, can nullify the benefits of parallelization and the power of the distributed computing resources. Moreover, not only the performance of the parallel application is deteriorated, but also issues related to energy consumption are affected if the problem of scheduling is not properly handled [17, 20, 28].

Johnatan E. Pecero · Pascal Bouvry
University of Luxembourg
E-mail: {johnatan.pecero, pascal.bouvry}@uni.lu

Aurelio A. Santiago Pineda · Héctor J. Fraire Huacuja · Juan J. Gonzalez Barbosa
Instituto Tecnológico de Ciudad Madero
E-mail: alx.santiago@gmail.com, automatas2002@yahoo.com.mx
E-mail: jjgonzalezbarbosa@hotmail.com

In its general form, the precedence-constrained scheduling problem is NP-complete [7]. Therefore, many heuristics based scheduling algorithms have been proposed that find a sub-optimal solution and attempt to balance running time, complexity and schedule quality [5, 14–16, 19, 25]. However, the performance of these heuristics is still an open research problem [4, 10, 11, 27, 31]. Consequently, there is an increasing interest in investigating and designing heuristics for scheduling precedence-constrained parallel programs on heterogeneous computing systems. Main motivations are not only the availability of heterogeneous computing platforms, such as Grid and P2P systems, but also the increasing interest in industry and science by executing a number of parallel applications that can be modelled by precedence task graphs or workflows [11].

In this paper we propose a local search algorithm based on an iterative search process to solve the precedence-constrained scheduling problem. The iterative local search (ILS) algorithm is a straight-forward, yet powerful techique for extending simple local search algorithms. We generate a benchmark composed of small size synthetic deterministic parallel program graphs proposed in literature. We compute the optimal solution by an enumerative search process that exhaustively explores the search space. We compare the performance of the proposed ILS algorithm with the optimal values considering an approximation factor. To evaluate and investigate scalability issues we also simulated the ILS algorithm using parallel graphs that model real-world applications. Results of the experimental study show that the algorithm is efficient in solving the problem providing results close to the optimal value.

The paper is organized as follows. In Section 2 we describe the precedence-constrained scheduling problem. Section 3 discusses related work. The proposed iterated local search algorithm is described in Section 4. Next, in Section 5 we present the benchmark and experimental results. Section 6 concludes the paper.

2 Problem description

The target system used in this work is represented by an undirected unweighted graph $G_s = (V_s, E_s)$, called a system graph (see, e.g. [31]). V_s is the set of N_s nodes of the system graph representing the m processors. E_s is the set of edges representing bidirectional channels between processors and defines a topology of the distributed system. The processors have different processing speed or provide different processing performance in term of MIPS (Million Instruction Per Second) and communication via links does not consume any processor time.

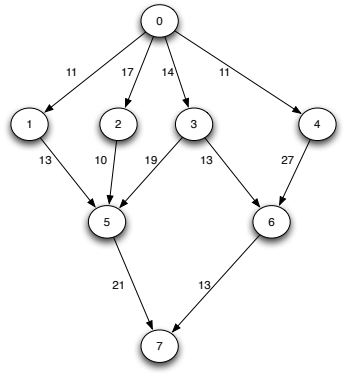
As usually, a parallel program is represented by a weighted directed acyclic graph (DAG). The DAG, called a precedence task graph or a program graph, is defined as $G = (T, E)$, where T is a finite set of nodes (vertices) and E is a finite set of edges. The node $t_i \in T$ is associated with one task t_i of the modeled parallel program. To every task t_i, there is an associated value p_{ij} representing the computation cost of the task t_i on a processor m_j, and its average computation cost is denoted as $\overline{p_i}$. Each edge $(t_{i_1}, t_{i_2}) \in E$ (with $t_{i_1}, t_{i_2} \in T$) is a precedence constraint between tasks and represents inter-task data communications if the output produced by task t_{i_1} has to be communicated to the task t_{i_2}. We consider the same communication model as in [4]. That is, the data parameter is a $t \times t$ matrix of communication data, where $data(t_i, t_j)$ is the amount of data required to be transmitted from task t_i to task t_j. The *rate* parameter is a $m \times m$ matrix and represent the data transfer rate between processors. The communication cost of edge $(t_i, t_j) \in E$, which is for data transfer from task

t_i (scheduled on processor m_k) to task t_j (scheduled on processor m_l), is defined by $c_{t_i,t_j} = data(t_i,t_j)/rate(m_k,m_l)$. When both t_i and t_j are scheduled on the same processor ($p_k = p_l$), then c_{t_i,t_j} becomes zero. The average communication cost of an edge is defined by $\overline{c_{t_i,t_j}} = data(t_i,t_j)/\overline{rate}$, where $\overline{rate}$ is the average transfer rate between the processors in the domain. For a given DAG the communication to computation ratio (CCR) is a measure that indicates wether a task graph is communication intensive, computation intensive or moderate. It is computed by the average communication cost divided by the average computation cost on a target system.

A simple task graph with its details are shown in Figure 1. The values presented in the last column of the table are computed based on a frequently used task prioritization method, the bottom level (*blevel*). The blevel of a node is the length of the longest path from the node to an exit node. Note that, both the computation and communication costs are averaged over all nodes and links. The $blevel(t_i)$ is computed recursively by traversing the DAG upward starting from the exit task t_{exit} as follows (Eq. 1):

$$blevel(t_i) = \overline{p_i} + max_{t_j \in succ(t_i)}\{blevel(t_j) + c_{ij}\}, \tag{1}$$

where $succ(t_i)$ is the set of immediate successors of t_i and $blevel(t_{exit}) = (\overline{p_{t_{exit}}})$.



task	r_0	r_1	r_2	$\overline{p_i}$	blevel
0	11	13	9	11	101.3
1	10	15	11	12	66.7
2	9	12	14	12	63.3
3	12	16	10	12	73.0
4	15	11	19	15	79.3
5	13	9	5	9	41.7
6	11	15	13	12	37.3
7	11	15	10	12	12.0

Fig. 1 In the left a sample DAG with the task indexes i inside nodes and values of $c_{t_{i_1}t_{i_2}}$ function next to the corresponding edges. In the right computation cost (p_i at level L_0) and task priorities (blevel).

The aim of scheduling is to distribute the tasks among the processors in such a way that the precedence constraints are preserved, and the response time C_{max} (the total execution time or makespan) is minimized. The response time C_{max} for a given precedence task graph depends on the allocation of tasks in the distributed computing topology and scheduling policy applied in individual processors [31]:

$$C_{max} = f(allocation, scheduling_policy) \tag{2}$$

The scheduling policy defines and order of processing tasks and assigns a starting time for each task, ready to run in a given processor. We will assume that it is the same for any run of the scheduling algorithm and we will focus on looking for such an allocation of tasks of a parallel application in a distributed computing system to

minimize the makespan. Although the minimization of makespan is crucial, tasks of a DAG in this work are not associated with deadlines as in real-time systems.

An approximation factor is used to evaluate the proposed algorithms. The factor is defined as $\rho = \frac{C_{max}}{C^*_{max}}$, where C^*_{max} is the optimal response time or makespan [11].

3 Related Work

The scheduling problem is NP-hard in its simplest version (homogeneous case without considering communications). Therefore, many heuristics have been proposed to schedule DAG applications on heterogeneous distributed computing systems. A well known scheduling algorithm is the Heterogeneous Earliest Finish Time (HEFT) algorithm [33]. The HEFT algorithm maintains a list of all tasks of a given graph according to their priorities, usually based on the blevel method. It consists in two phases. In the first phase, a ready task is selected from the priority list. The task with the highest priority for which all dependent tasks have finished is chosen. This process corresponds to the *task prioritizing* or *task selection* phase. Thereafter, a suitable processor that minimizes a predefined cost function is selected (i.e., the *processor selection* phase), in this case the processor which will result in the earliest finish time of that task. The HEFT algorithm is one of the most used algorithm as a basis for comparison to evaluate the performance of new proposed scheduling algorithms [2–5, 8–10, 12, 13, 15, 16, 25–27, 29, 30, 35]. Therefore, we also use HEFT in Section 5 to validate the proposed approach.

A number of local search algorithms for scheduling have been investigated in the literature. Kowk et al. [24] present a first improvement random local search algorithm, named FAST. In this algorithm, a task is randomly picked and then moved to a randomly selected processor. If the schedule length is reduced, the move is accepted. Otherwise, the task is moved back to its original processor. Kwok and Ahmad in [23] modified the FAST algorithm. The major improvement is that it uses a nested loop for a probabilistic jump. A parallel version of FAST is named FASTEST. Wu et. al. in [34] proposed a local search algorithm based on topological ordering. The algorithm is a deterministic guided search that uses the level of a task, defined as the sum of the top-level and blevel, to schedule the tasks. The algorithm first selects a task with the largest level and then assigns it to the processor that generates the smallest level for that task. The level for each task is dynamically calculated and is used to determine the search direction. However, the considered computing system is based on homogeneous processors. Kim et al. [18] report a deterministic local search-based scheduling algorithm. The algorithm starts with a schedule found by a deterministic scheduling algorithm and then iteratively attempts to improve the current best solution using a deterministic guided search method based on prior knowledge about the task scheduling problem and the target computing environment. The main idea is to move tasks to fill the idle periods of processors. One of the major limitations of this approach is that it assumes complete knowledge of the problem and that the information about tasks and communications is always accurate. However, many external events can modify the parameters of the scheduling problem. Kang et al. [14] propose an iterated greedy algorithm. The main idea of this algorithm is to improve the quality of the assignment in an iterative manner using results from previous iterations. The algorithm first uses a constructive heuristic to find an initial assignment and iteratively improves it in

a greedy way. The authors consider additional resource constraints in the scheduling problem. However, in this work we do not consider such constraints.

4 Iterated Local Search Based Scheduling

The solution we propose in this work is a scheduling algorithm based on a local search. Local search was one of the early techniques for combinatorial optimization. The principle is to refine a given initial solution point in the solution space by searching through a neighborhood of the solution point. If an improvement can be achieved in this manner, then a new solution is obtained. This process is continued until no further improvement can be obtained. However, in many cases, local minima are common and local search algorithms can converge quickly to these local minima and get stuck in a local optimum solution far away from the global optimal. We propose a local search algorithm based on an iterated local search (ILS) to alleviate this problem.

Algorithm 1 shows the pseudocode of ILS we have used in this paper. The ILS algorithm is a trajectory-based metaheuristic that can be seen as a straight-forward, yet powerful technique for extending simple local search algorithms. The algorithm starts off by generating and evaluating an initial solution. The search process (i.e., the initial solution) can be initialized in various ways, for example, by starting from a randomly generated initial solution or from a heuristically constructed solution to the given problem. Then, following an iteration based approach, it seeks to improve the solutions from one iteration to the next. At each iteration, a perturbation of the obtained local optimum is carried out. The perturbation mechanism introduces a modification to a given candidate solution to allow the search process to escape from local optimum. A local search is applied to the perturbed solution. The new solution is then evaluated and accepted as the new current solution under some conditions. The algorithm finishes when the termination condition is met. We will detail in the following subsections the core components of the ILS customized to the task scheduling problem.

Algorithm 1 Algorithm outline of ILS based scheduling algorithm

```
 1: sol = GenerateInitialSolution();
 2: EvaluateSolution(sol);
 3: bestSol = sol;
 4: repeat
 5:    Perturbation(sol);
 6:    LocalSearch(sol);
 7:    EvaluateSolution(sol);
 8:    if sol < bestSol then
 9:       bestSol = sol;
10:    end if
11: until termination condition met
```

4.1 Initial solution

The ILS based scheduling algorithm starts of by generating an initial feasible solution as the starting point of search by the local search procedure. We have constructed the

initial solutions as follows. First, we sort tasks by priority, then we select the task with highest priority and schedule the task to the processor that optimizes a predetermined objective function. To assign priorities, two different methods have been evaluated, the first one based on the blevel method and the second one is based on list scheduling principle using random feasible orders. The main idea of random feasible orders is to randomly select a task among the ready tasks and place that task in the top of a priority list. A task is ready when it is an entry task or when all its predecessors have already been selected and are in the priority list. The tasks are selected from the priority list and scheduled on the basis of the HEFT heuristic, that is assigning the task to the processor that minimizes the earliest finish time. In the case of blevel + HEFT, the quality of the initial solution is equal to that of the HEFT algorithm. In case of random feasible orders the quality of the solution depends on the priority of the tasks. In this case, we generate 50 initial random feasible solutions. We evaluate each of the generated solutions and we keep the best among them. Hence, we have evaluated two different methods of constructing the initial solution.

4.2 Perturbation process

The perturbation process is an essential aspect determining the good behavior of an ILS algorithm, as well as the local search process.

We have tested two different configurations for the perturbation method. As that, the search space is differently explored. In the first perturbation process, called *probability based movement*, every task t_i of the current solution has a probability to be changed from its current processor. If the probability occurs the task t_i will be moved from its current location and assigned to a new random processor. The second perturbation process, called *random movement* moves only one task at a time from its current location to a random processor. Although the new assignment in the perturbation methods can be selected by a sophisticated criterion, we have decided to randomly select the new processor to keep time complexity of the algorithm as low as possible.

4.3 Local search

We propose a local search based on the *first improvement* type pivoting rule. The first improvement strategy tries to avoid the time complexity of evaluating all neighborhoods by performing the first improvement step encountered during the inspection of the neighborhood. The proposed algorithm evaluates the neighboring candidate solutions in a particular fixed order. We use the order of the priority list computed in the initial solution construction (i.e., blevel or random feasible order) as classical list scheduling algorithms. For each task in the list the neighborhood is defined to be the set of assignments that can be obtained by removing the task from its current processor and reallocating it to another. If a makespan improvement is found, then the search process is restarted from the new solution.

4.4 Acceptance criterion and termination condition

An important step in the ILS algorithm is to consider whether the new schedule is accepted or not as the incumbent solution for the next iteration. The proposed ILS accepts solutions if and only if the makespan is improved, and rejects them otherwise.

Different conditions can be used to stop the algorithm. The ILS based scheduler stops the process when the algorithm reaches a maximum number of iterations.

5 Synthetic Benchmark and Performance Comparison

Many heuristics have been developed to solve the heterogeneous DAG scheduling problem. Most of them use HEFT as a basis for comparison because of the lack of generally accepted standard benchmarks for the evaluation of scheduling heuristics in the heterogeneous distributed computing systems. In this work, we construct a benchmark as follows. We have collected a set of small size synthetic deterministic non-preemptive program graphs from literature. We compute the optimal solution and the optimal value by an enumerative algorithm that performs an exhaustive search. The algorithm explores all the possible solutions in the search space by keeping the best found solution. The problem is reduced to generate possible permutations with and without repetitions and evaluate the resulting solutions.

Since generating optimal solutions for arbitrarily structured task graphs takes exponential time, it is not feasible to obtain optimal solutions for larger graphs [22]. However, to investigate the scalability of the proposed ILS algorithm we have also used a set of structured real-world parallel applications. The applications used for our experiments are the robot control application and sparse matrix solver from the Standard Task Graph set (STG) [32], and a subroutine of the Laser Interferometer Gravitational-wave Observatory (LIGO) application [6]. Table 1 summarizes the main characteristics for the used applications: instances size, edges amount and the ratio between tasks and edges (ETR). ETR gives information of the degree of parallelism.

Table 1 Instance types: tasks and edges numbers, and Edge Task Ratio.

Type	Tasks	Edges	ETR
LIGO	76	132	1.73
Robot	88	131	1.48
Sparse	96	67	0.69

The STG set is composed of homogenous instances with constant execution time among machines. Since we are interested in heterogenous instances, we have only considered the structure of these applications and we have implemented the procedure described in [33] to consider heterogeneity. We fixed the parameter β to 1. Parameter β is basically the heterogeneity factor for processor speeds. A high percentage value (i.e., a percentage of 1) causes a significant difference in a task's computation cost among the processors. For each graph we have varied the CCR ratio. A randomization procedure which changes weight of edges was executed to assure the needed CCR. We have generated five CCRs (0.1, 0.5, 1, 5, 10) for each graph. Tested system sizes were 8, 16 and 32 processors. We have generated 15 different instances for each application type.

5.1 ILS Configurations

We have four different ILS configurations depending on the method used to generate the initial solution and the perturbation process. The four ILS algorithms studied are listed below.

- **ILS-Alg1**: It uses the blevel method to generate the order of tasks' execution and generate the initial solution. The perturbation process is the probability based movement.
- **ILS-Alg2**: The initial solution is based on the blevel method and the used perturbation process is the random movement.
- **ILS-Alg3**: The algorithm generates the initial solutions based on the random feasible orders. The algorithm generates 50 solutions and the best one is used as the initial solution. The perturbation process is the probability based movement.
- **ILS-Alg4**: To construct the initial solution the algorithm generates 50 solutions using the random feasible orders method and keep the best among all the generated solutions. The algorithm applies the random movement to perturb the solution during the iteration process.

We study in Figure 2 the effects of different probabilities of the perturbation process based on the probability movement in order to identify the probability to set in the experiments. For this experiment we used a LIGO instance in an eight processors system, and CCR equal to 5. Figure 2 shows a typical run of the algorithms for different probabilities from 0.05 to 0.2 with 0.05 increment. We only show the results obtained by ILS-Alg1, however the results for the rest of algorithms show the same behavior. The main result that can be drawn from this study is that probabilities equal to 0.05 and 0.15 perform better than the other two probabilities (equal to 0.1 and 0.2). When comparing these two probabilities we noticed that using a low probability (0.05) leads to a faster convergence. Therefore the results we report for the algorithms ILS-Alg1 and ILS-Alg3 were computed using that probability movement. The maximum number of iterations in the local search process without improving the current best solution was set to 50. The maximum number of iterations per algorithm (i.e., the termination condition) was set to 15. The algorithms are executed 15 times on each instance and each run is independent.

5.2 Results for Synthetic Deterministic Program Graphs

We report in this section some preliminary results obtained with the ILS algorithms. We compare the results against the global optimal value computed with the enumerative search method. We also compare the results obtained by HEFT. Table 2 shows the results for the results reported in literature and the optimal value. The first column corresponds to the name of the instance. The name of the instance is composed as follows: *id_m_n_bestres*, where id represents the surname of the main author of the reference where the instances are proposed, m is the number of processors used in the reference to schedule the instance, n is the size of the instances, and *bestres* corresponds to the best value reported in literature. For example, instance *Ahmad_3_9_28* is read as: Ahmad the id of the instance simulated on three processors system, with 9 tasks in size and the best result reported is equal to 28 units of time. The second column in Table 2 provides the reference where the instance is proposed. The third column presents the

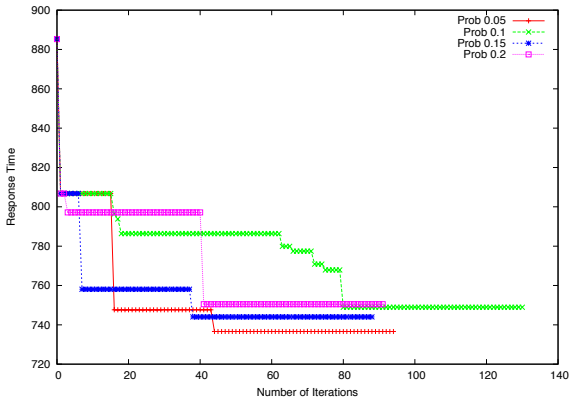


Fig. 2 Results for various probability values used by the perturbation process based on probability movement

Table 2 Results for synthetic deterministic program graphs known in the literature and the optimal value

Instance	*Reference*	*Best Result*	*optimal*
Ahmad_3_9_28	[1]	28	22
Hsu_3_10_84	[12]	84	80
Eswari_2_11_61	[10]	61	56
Hamid_3_10	[2]	?	100
Heteropar_4_12_124	[3]	124	124
Ilavarasan_3_10_77	[13]	77	73
Kang_3_10_76	[16]	76	73
Kang_3_10_84	[15]	84	79
Kuan_3_10_28	[25]	28	26
Liang_3_10_80	[26]	80	73
Daoud_2_11_64	[8]	64	56
YCLee_3_8_80	[27]	80	66
Sample_3_8_100		100	81
*SampleFig*1_3_8_89		89	66

best result reported in literature. The fourth column shows the optimal value computed with the enumerative search algorithm. Table 3 depicts the results computed by HEFT in the third column. The last four columns present the best response time computed by each one of the four ILS configurations and the number of times over 15 independent runs that the best value is equal to the optimal value. Rows in gray color highlight the algorithm that is able to find the optimal value for a given instance in all the executions.

As we can notice in Table 2 the optimal solutions can not be always found by the algorithms despite having small size instances with maximum four processor system and 12 tasks. Moreover, there are some instances for which the ILS configurations are not able to find the optimal solutions in each run. Recall that the scheduling problem is NP-hard even for two processors system.

From Table 3 it can be observed that the ILS configurations with initial solutions generated using random feasible orders (i.e., ILS_Alg3 and ILS_Alg4) find the optimal

Table 3 Comparison results for synthetic deterministic program graphs known in the literature

Instance	*optimal*	HEFT	ILS_Alg1	ILS_Alg2	ILS_Alg3	ILS_Alg4
Ahmad_3_9_28	22	24	23 (0)	23 (0)	22 (15)	22 (15)
Hsu_3_10_84	80	92	80 (12)	80 (13)	80 (14)	80 (15)
Eswari_2_11_61 61	56	76	58 (0)	58 (0)	56 (4)	56 (2)
Hamid_3_10	100	110	100 (10)	100 (10)	100 (15)	100 (15)
Heteropar_4_12_124	124	150	124 (12)	124 (15)	124 (15)	124 (15)
Ilavarasan_3_10_77	73	80	73 (13)	73 (13)	73 (10)	73 (10)
Kang_3_10_76	73	80	75 (0)	75 (0)	73 (5)	73 (4)
Kang_3_10_84	79	109	82 (0)	82 (0)	83 (0)	83 (0)
Kuan_3_10_28	26	30	26 (15)	26 (15)	26 (8)	26 (11)
Liang_3_10_80	73	80	73 (13)	73 (13)	73 (15)	73 (10)
Daoud_2_11_64	56	76	58 (0)	58 (0)	56 (3)	56 (1)
YCLee_3_8_80	66	88	68 (0)	68 (0)	66 (15)	66 (15)
Sample_3_8_100	81	84	81 (1)	81 (7)	81 (4)	81 (4)
*SampleFig*1_3_8_89	66	89	69 (0)	69 (0)	66 (15)	66 (15)

value for a bigger number of instances than the ILS algorithms using blevel as initial solution (i.e., ILS_Alg1 and ILS_Alg2). The main reason is that by generating random initial solutions the algorithm is able to explore different regions of the search space. However, using the deterministic blevel method the algorithms are restricted to a specific region of the search space. In fact, the ILS_Alg3 and ILS_Alg4 algorithms find the optimal value in around 93% of the instances and the algorithms ILS_Alg1 and ILS_Alg2 find the optimal value for only 56% of the instances. The ILS configurations with random feasible orders not only find the optimal value for a bigger number of instances than ILS configurations with blevel, but also are able to find optimal values in all the executions for a bigger number of instances than ILS_Alg1 and ILS_Alg2. It can be verified in Table 3 with rows highlighted in gray color.

Table 4 Approximation Factor

Instance	HEFT	ILS_Alg1	ILS_Alg2	ILS_Alg3	ILS_Alg4
Ahmad_3_9_28	1.090	1.045	1.045	1	1
Hsu_3_10_84	1.150	1.012	1.011	1.001	1
Eswari_2_11_61	1.366	1.061	1.038	1.054	1.064
Hamid_3_10	1.100	1.019	1.019	1	1
Heteropar_4_12_124	1.209	1.036	1	1	1
Ilavarasan_3_10_77	1.095	1.005	1.005	1.012	1.013
Kang_3_10_76	1.095	1.030	1.030	1.025	1.026
Kang_3_10_84	1.379	1.043	1.046	1.058	1.049
Kuan_3_10_28	1.153	1	1	1.028	1.015
Liang_3_10_80	1.095	1.005	1.005	1	1.010
Daoud_2_11_64	1.366	1.057	1.042	1.058	1.056
YCLee_3_8_80	1.333	1.030	1.030	1	1
Sample_3_8_100	1.037	1.018	1.008	1.012	1.009
*SampleFig*1_3_8_89	1.348	1.045	1.045	1	1
Average	1.191	1.029	1.022	1.017	1.018

Table 4 provides more details about the performance comparison by presenting the approximation factor (ρ) for each algorithm. First column provides the name of instances. Second column shows the approximation factor for the HEFT algorithm. Recall that HEFT is a well-known scheduling algorithm and is often used as a basis for comparison and validation of new proposed heuristics. However, to the best of our knowledge no comparison has been provided between the results computed by HEFT and the optimal value. The remainder columns present the approximation factor for the ILS-configurations. We provide average approximation factor for the ILS algorithms. From Table 4 we observe that the ILS algorithm with the different configurations outperform HEFT. The ILS algorithms are around 3% on average (for ILS_Alg1) from the optimal values, however HEFT is around 19% on average from the optimal values. We see that ILS configurations based on random feasible orders compute best results than ILS algorithms using blevel to generate the initial solution. However, the results are comparable among the four configurations.

5.3 Results for Real-world Parallel Applications

In this section we present results using the set of real-world parallel applications. The aim is to evaluate the scalability of the ILS configurations. We present statistical results. In our test, the algorithms were run for 15 independent runs. We have evaluated the ILS algorithms using a Friedman test [21]. The Friedman test is a non-parametric statistical tool that allows to compare a set of a non normalized population to verify if does exist significant statistical differences in the sample. Moreover, the ranking shows which algorithm has the higher performance.

Table 5 provides the results obtained after applying the Friedman test. The ranking shows the performance of the algorithms. As we are dealing with a minimization problem, the higher the value of the ranking, the better the algorithm performs in finding a solution for the set of instances. We can observe that the behavior of the algorithms is different when the problem scale regarding small size instances. The rank shows that ILS-Alg4 is still computing best results than the other three configurations. However, ILS-Alg2 shows better performance than ILS-Alg1 and ILS-ALg3. The common characteristics of both algorithms is that ILS-Alg4 and ILS-Alg2 use the same perturbation process, which slightly change the current solution by moving only one task at a time then exploring the neighbors of the perturbed solution. The main reason is that the changes made by the algorithms (moving only one task) do not violate a structure of the population in a huge range. At once the algorithm can continuously search better and better solution [31]. However, it can be interesting to investigate a more sophisticated heuristic to decide which task to move based on some knowledge of the current structure of the scheduling problem under consideration without increasing complexity. We can assume that the ILS algorithm provides results better or comparable than HEFT. We can verify the results by considering that ILS-Alg1 and ILS-Alg2 use as initial solution the output of HEFT, hence if ILS does not improve the initial solution we have the same results than HEFT.

We performed a statistical hypothesis testing to verify if the difference between the algorithm that provides the best results are significant if the null hypothesis is rejected, or otherwise, if due to the random nature of the solutions, the obtained results may be considered statistically equivalent.

Table 5 Average Rankings of the algorithms on all the set of instances

Algorithm	LIGO	Robot	Sparse
ILS-Alg1	2.2888	2.3733	2.1155
ILS-Alg2	3.0466	2.9333	3.2355
ILS-Alg3	1.3444	1.3000	1.6155
ILS-Alg4	**3.3200**	**3.3933**	**3.0333**

Table 6 presents the statistical tests that take into account the p-value obtained between all the algorithms with the best performing one (ILS-Alg4), with a level of significance of $\alpha = 0.05$, if the observed value is larger than $1 - \alpha$ quantile of this distribution, the null hypothesis is rejected. As mentioned, the results indicates that at least the candidate algorithm gives better performance than at least one of the compared algorithms.

In Table 6 we can observe that ILS-Alg4 outperforms ILS-Alg1 and ILS-Alg3 configurations, but is statistically equivalent to ILS-Alg2. Meanwhile, ILS-Alg4 finds better solutions than ILS-Alg2 when considering Robot application.

Table 6 Statistical hypothesis testing for the best algorithm by Makespan with $\alpha = 0.05/4$ (Bonferroni correction)

Set	ILS-Alg4 vs. Algorithm	p-value	H_0 Rejected
LIGO	ILS-Alg1	2.4237E-17	Yes
	ILS-Alg2	0.0247	No
	ILS-Alg3	3.0522E-59	Yes
Robot	ILS-Alg1	5.2857E-17	Yes
	ILS-Alg2	1.5727E-4	Yes
	ILS-Alg3	2.7254E-66	Yes
Sparse	ILS-Alg1	4.6884E-14	Yes
	ILS-Alg2	0.0966	No
	ILS-Alg3	2.3426E-31	Yes

6 Conclusions and Future Work

In this paper we have proposed an iterative local search based scheduling algorithm to solve the precedence-constrained scheduling problem in heterogeneous computing systems. We have generated a synthetic benchmark composed of small size deterministic parallel applications. We computed the optimal solutions by using an exhaustive enumerative search algorithm. The generated benchmark can be used as a basis for comparison during the design of new scheduling heuristics.

We have investigated four configurations for the proposed ILS algorithm. We compared the results against optimal solutions. From the set of experimental results we observed that ILS benefited not only from the generation of several feasible orders (diversifying solutions) and considering the best one instead of generating only one solution, but also from the small perturbations in the current solution intensifying the search in its neighborhood, two basic considerations when designing local search methods. One of the main advantages of the propose ILS is that it can be coupled with a more sophisticated heuristic without increasing complexity.

We plan to extend the proposed work. First, we consider to extend the benchmark. For that, we plan to parallelize the enumerative search method and design a branch and bound algorithm. We consider to design more sophisticated heuristics to decide which task to move during the perturbation process. One possibility to explore is to move the task with highest CCR, such that the communication delay can be reduced. Nowadays, one important aspect to consider is the size of the workflows which usually are composed of thousands of tasks, we plan to evaluate the scalability of proposed local search in bigger workflows, for that, we consider to apply a partitioning technique to first decompose the DAG graph in subgraphs, and then apply our local search in a cooperative way to locally optimizes these subgraphs, while considering the global optimization.

Acknowledgements This work is partially supported by the National Research Fund, Luxembourg in the framework of the AFR Green Energy-Efficient Computing project (PDR-09-067). The mexican researchers was supported by the Consejo Nacional de Ciencia y Tecnologia (CONACyT), Mexico.

References

1. Ahmad, I., Dhodhi, M., UI-Mustafa, R.: Dps: dynamic priority scheduling heuristic for heterogeneous computing systems. IEE Proceedings: Computers and Digital Techniques **145**(6), 411–418 (1998)
2. Arabnejad, H.: List based task scheduling algorithms on heterogeneous systems - an overview. http://paginas.fe.up.pt/prodei/dsie12/papers/paper_30.pdf (2011). Available Online. Consulted January, 2013
3. Arabnejad, H., Barbosa, J.: Performance evaluation of list based scheduling on heterogeneous systems. http://icl.eecs.utk.edu/heteropar2011/slides/heteropar _JorgeBarbosa.pdf. HeteroPar'11. Consulted online 2012
4. Arabnejad, H., Barbosa, J.G.: Performance evaluation of list based scheduling on heterogeneous systems. In: Proceedings of the 2011 international conference on Parallel Processing, Euro-Par'11, pp. 440–449. Springer-Verlag, Berlin, Heidelberg (2012)
5. Bittencourt, L.F., Sakellariou, R., Madeira, E.R.M.: Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In: Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, PDP '10, pp. 27–34. IEEE Computer Society (2010)
6. Brown, D., Brady, P., Dietz, A., Cao, J., Johnson, B., McNabb, J.: A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis. In: I. Taylor, E. Deelman, D. Gannon, M. Shields (eds.) Workflows for e-Science, pp. 39–59. Springer London (2007)
7. Coffman, E.G.: Computer and jobshop scheduling theory. John Wiley & Sons Inc (1976)
8. Daoud, M.I., Kharma, N.: A hybrid heuristic-genetic algorithm for task scheduling in heterogeneous processor networks. J. Parallel Distrib. Comput. **71**(11), 1518–1531 (2011)
9. Demiröz, B., Topcuoglu, H.R.: Static task scheduling with a unified objective on time and resource domains. Comput. J. **49**(6), 731–743 (2006)
10. Eswari, R., Nickolas, S.: Path-based heuristic task scheduling algorithm for heterogeneous distributed computing systems. In: Proceedings of the 2010 International Conference on Advances in Recent Technologies in Communication and Computing, ARTCOM '10, pp. 30–34. IEEE Computer Society, Washington, DC, USA (2010)
11. Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., González-García, J., Röblitz, T., Ramírez-Alcaraz, J.: Multiple workflow scheduling strategies with user run time estimates on a grid. J. Grid Comput. **10**(2), 325–346 (2012)
12. Hsu, C.H., Hsieh, C.W., Yang, C.T.: A generalized critical task anticipation technique for dag scheduling. In: Proceedings of the 7th international conference on Algorithms and architectures for parallel processing, ICA3PP'07, pp. 493–505. Springer-Verlag, Berlin, Heidelberg (2007)

13. Ilavarasan, E., Thambidurai, P., Mahilmannan, R.: Performance effective task scheduling algorithm for heterogeneous computing system. In: Proceedings of the The 4th International Symposium on Parallel and Distributed Computing, ISPDC '05, pp. 28–38. IEEE Computer Society, Washington, DC, USA (2005)
14. Kang, Q., He, H., Song, H.: Task assignment in heterogeneous computing systems using an effective iterated greedy algorithm. J. Syst. Softw. **84**(6), 985–992 (2011)
15. Kang, Y., Lin, Y.: A recursive algorithm for scheduling of tasks in a heterogeneous distributed environment. In: Y. Ding, Y. Peng, R. Shi, K. Hao, L. Wang (eds.) BMEI, pp. 2099–2103. IEEE (2011)
16. Kang, Y., Zhang, Z., Chen, P.: An activity-based genetic algorithm approach to multiprocessor scheduling. In: Y. Ding, H. Wang, N. Xiong, K. Hao, L. Wang (eds.) ICNC, pp. 1048–1052. IEEE (2011)
17. Khan, S.U., Ahmad, I.: A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids. IEEE Trans. Parallel Distrib. Syst. **20**(3), 346–360 (2009)
18. Kim, S.C., Lee, S., Hahm, J.: Push-pull: Deterministic search-based dag scheduling for heterogeneous cluster systems. IEEE Trans. Parallel Distrib. Syst. **18**(11), 1489–1502 (2007)
19. Kolodziej, J., Khan, S.U.: Multi-level hierarchic genetic-based scheduling of independent jobs in dynamic heterogeneous grid environment. Inf. Sci. **214**, 1–19 (2012)
20. Kolodziej, J., Khan, S.U., Xhafa, F.: Genetic algorithms for energy-aware scheduling in computational grids. In: F. Xhafa, L. Barolli, J. Kolodziej, S.U. Khan (eds.) 3PGCIC, pp. 17–24. IEEE (2011)
21. Kvam, P.H., Vidakovic, B.: Nonparametric Statistics with Applications to Science and Engineering (Wiley Series in Probability and Statistics). Wiley-Interscience (2007)
22. Kwok, Y.K., Ahmad, I.: Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm. J. Parallel Distrib. Comput. **47**(1), 58–77 (1997). DOI 10.1006/jpdc.1997.1395. URL http://dx.doi.org/10.1006/jpdc.1997.1395
23. Kwok, Y.K., Ahmad, I.: Fastest: A practical low-complexity algorithm for compile-time assignment of parallel programs to multiprocessors. IEEE Trans. Parallel Distrib. Syst. **10**(2), 147–159 (1999)
24. Kwok, Y.K., Ahmad, I., Gu, J.: Fast: A low-complexity algorithm for efficient scheduling of dags on parallel processors. In: ICPP, Vol. 2, pp. 150–157 (1996)
25. Lai, K.C., Yang, C.T.: A dominant predecessor duplication scheduling algorithm for heterogeneous systems. J. Supercomput. **44**(2), 126–145 (2008)
26. Lee, L.T., Chen, C.W., Chang, H.Y., Tang, C.C., Pan, K.C.: A non-critical path earliest-finish algorithm for inter-dependent tasks in heterogeneous computing environments. In: HPCC, pp. 603–608. IEEE (2009)
27. Lee, Y.C., Zomaya, A.: A novel state transition method for metaheuristic-based scheduling in heterogeneous computing systems. IEEE Trans. Parallel Distrib. Syst. **19**(9), 1215–1223 (2008)
28. Lee, Y.C., Zomaya, A.Y.: Energy conscious scheduling for distributed computing systems under different operating conditions. IEEE Trans. Parallel Distrib. Syst. **22**(8), 1374–1381 (2011)
29. Liu, G.Q., Poh, K.L., Xie, M.: Iterative list scheduling for heterogeneous computing. J. Parallel Distrib. Comput. **65**(5), 654–665 (2005)
30. Shen, L., Choe, T.Y.: Posterior task scheduling algorithms for heterogeneous computing systems. In: Proceedings of the 7th international conference on High performance computing for computational science, VECPAR'06, pp. 172–183. Springer-Verlag (2007)
31. Switalski, P., Seredynski, F.: Multiprocessor scheduling by generalized extremal optimization. J. of Scheduling **13**(5), 531–543 (2010)
32. Tobita, T., Kasahara, H.: A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. Journal of Scheduling **5**(5), 379–394 (2002)
33. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
34. Wu, M.Y., Shu, W., Gu, J.: Efficient local search for dag scheduling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 617–627 (2001)
35. Zhao, H., Sakellariou, R.: An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In: H. Kosch, H. Böszörményi Lászlóand Hellwagner (eds.) Euro-Par 2003 Parallel Processing, *Lecture Notes in Computer Science*, vol. 2790, pp. 189–194. Springer Berlin / Heidelberg (2003)

MISTA 2013

Scheduling of electric vehicle charging operations

Damir Bucar, · Sandford Bessler, · Nysret Musliu, · Jesper Groenbaek

Abstract In this work we address the optimization problem related to the recharge of electric vehicles, in which parking place availability, parking duration and available power constraints have to be satisfied. We formulate this problem as a resource allocation problem (with time windows) and solve it with exact and heuristic methods. The results obtained with a min-conflicts heuristic are promising. Further, we apply the static problem to an online, real world environment, and shortly discuss additional challenges and the performance of the overall system.

1 Introduction

The electric vehicle (EV) charging is a topic that draws the attention of many scientific communities: their members are power and smart grid experts, signal processing and telecommunications scientists, mobility as well as traffic engineers, or computer scientists specialized in algorithms. Electromobility is indeed at the crossroads of several disciplines and innovation is still possible, as a single standardized solution is not yet deployed or even agreed upon.

As the recharging power doubles the consumption of households, a massive deployment of charging operations will affect in particular the low voltage grids. With today's battery technology, EV charging durations spans still several hours. One way to improve the planning and customer satisfaction at public charging stations is to

Damir Bucar
Vienna University of Technology
E-mail: damir.bucar@gmail.com

Sandford Bessler
FTW Telecommunication Research Center
E-mail: bessler@ftw.at

Nysret Musliu
Vienna Univerity of Technology
E-mail: musliu@dbai.tuwien.ac.at

Jesper Groenbaek
FTW Telecommunication Research Center
E-mail: groenbaek@ftw.at

allow reservation during or before the trip. In this way, both a parking place (including a charging point) and the estimated energy required for charging can be planned well before the arrival of the car. Smart resource allocation algorithms are required in particular at public charging stations, because of the stochastic character of car arrivals and because of possible limitations in power and energy supply.

In the application domain, the problem is easily described: the EV parking lot owner (and charging service provider) has a list of vehicles, (some already arrived, the others on the way) with their planned parking duration and their required amount of energy. In addition, he knows how much power will be available for all the cars, in each time period, in the near future. The question is, when to switch on and off the charging power for every car, so that he can maximize a profit function?

A number of works have already shown that, optimized, grid-aware charging strategies reduce grid congestion and allow a higher penetration of EVs without additional investment in the grid infrastructure ([1], [3],[2], [4]). Most of the works exploit the so called "valleys" in the load curve of the local grid and shift the charging tasks into these periods. For charging at public stations, the model has however to include the estimated arrival at the charging station and the estimated parking time. A first contribution of this work is therefore to identify and formulate the EV charging problem as a resource allocation problem (RAP) with time intervals [10], [5], which is closely related to the bandwidth allocation problem (BAP) [13], and the resource constrained scheduling problem [12]. RAP is NP-hard, since it contains as special case the classical $0-1$ knapsack problem, where all time intervals are identical, and which is itself NP-hard.

Therefore, by using a mixed integer program and the CPLEX solver, we can solve problem instances smaller than 100 cars in reasonable time. For larger instances we investigated first greedy heuristics available in the literature, but with poor results. A more sophisticated heuristic, the local ratio, is investigated and is compared with a min-conflicts based heuristic, which performs best of all. We discuss the solution quality comparing it with the solution of the mixed integer program.

The rest of the paper is organized as follows: in Section 2 we formulate the EV charging problem as an integr program, in Section 3 we describe a number of heuristic solution approaches, in Section 4 we present computational results. In Section 5 we report on the extensions required for the online, real world scheduling version of the EV charging, and present first results. Finally in Section 6 we draw concluding remarks.

2 The EV charging problem

The EV charging problem which has been studied before [1], [2],[3], [4] is formulated in this work (for the first time, to our knowledge) as a machine scheduling problem with time windows. The machines are the charging points that charge the connected EVs with one of a few predefined charging speeds, and without preemption of the operation. Although preemption would add flexibility in scheduling, it would also add complexity in the handling of charging states and in the payment transactions. The time horizon is discretized in T discrete intervals (for instance 15 minutes each). Using the notation in [5], each EV is associated with an activity $a \in A$ that includes parking and charging at a charging point $m \in M$ (m includes the parking place). The actual charging has to occur within the given time window. The aim is to determine for each activity the charging start time, charging duration and speed, the allocation of activities to charging points,

Notation	Description
T	planning horizon with time slots of $t = 1..T$
A	Set of charging activities
M	set of charging points
S	set of charging speeds
I	instances generated from activities A
e_j, l_j	time window (earliest, latest) $j \in A$
s_i, f_i	start and finish time of charging , $i \in I$
$dmin_j$	minimum energy demand of activity j
$dmax_j$	maximum energy demand of activity j
a_i	activity that generates instance i
w_i	charging speed of instance i
p_i	calculated profit for instance i
P^t	total available power during $t \in T$
v_i^t	timeslots in which parking=true for instance i
x_i	variable: $x_i \in \{0, 1\}, i \in I$.
$k[w_i]$	cost for charging at speed w_i

Table 1 Notation summary

such that the total power consumed in any time period does not exceed a given value, and such that a certain profit objective function to be explained below is maximized.

Note that we have to deal with two types of resources: parking place during the whole time window, and power limitation during the charging period. If we make the assumption that the charging points are identical (machines), we can split the problem into two sub-problems:

- since the machines are identical, we allocate first the activity time windows to the charging points. This problem is similar to scheduling classes to classrooms, and can be solved optimally by a greedy heuristic (increasing starting time rule). The intervals are the activity time windows and cannot be shifted. The resulted allocation of activity to charging point allows us to address the second subproblem as a one machine problem.
- this is a bandwidth or resource allocation problem over time intervals, or RAP which is a NP-hard problem, since it can be reduced to a knapsack problem if the time windows are set to the interval [0,1]. The specific problem which we denote EVRSTW (electric vehicle recharge scheduling with time windows) requires that the charging takes place within the time window of the activity and is limited by total power available in each of the T periods.

In case the machines are not identical, i.e. are divided in two groups: slow and fast charging, the machine allocation can be still made under the following assumption: a fast charging point is an expensive resource, therefore is will not be used for slow charging, which leads to two disjoint activity groups. In the most general case, instances have to be created for each individual machine, making the problem very large.

In the rest of this section we assume the machines are identical, so that the allocation activity-charging point has been done and we focus only on the second subproblem, the RAP.

An important part of the model is the requested energy demand by an EV: a minimum demand *dmin* has to be satisfied and is calculated based on the state of charge on arrival at the charging station and the remaining trip in km until the next recharge. The maximum demand *dmax* is the energy required to fill the battery.

In order to deal with several resources and variables (such as charging speed) in resource allocation problems Bar-Noy et al [5] propose to create an "instance" for each combination of these variables. The original problem is solved when a subset of instances is selected. We use the same method to create an instance $i = i(a, m, w, d, l)$ for each combination of allowed values of the following variables: $a \in A$, $m \in M$, $w \in S$, $d \in [dmin_a/w, dmax_a/w]$, and $l \in [e_a, l_a - dur(d, w)]$, where the charging duration dur is computed from the speed and demand values.

After generating the set of instances I, the problem reduces to finding a set of binary variables x_i, $i \in I$, as we will see below. The profit of an instance p_i is defined as a weighted sum of two objective contributions: the completion degree (the ratio between the energy charged and the maximum energy requested) and the lifetime preservation factor k/w_i of the battery, which we assume to be inversely proportional to the charging speed, (k is a constant, e.g. k=2):

$$p_i = \alpha d_i/dmax_{a_i} + (1 - \alpha)k/w_i, 0 \leq \alpha \leq 1 \tag{1}$$

The profit of an activity is not uniform, making the search for a solution more difficult [10]. In Fig. 1 we depict the profit values of one activity. The function is discrete as the possible demands and the speeds build discrete sets, however some solution points are cut away by power constraints at high charging speeds and by the time window duration at lower speeds. When tuning the function p_i, we take care to assign the most weight to the completion degree, so that the preference for low charging speeds, only slightly affects the solution.

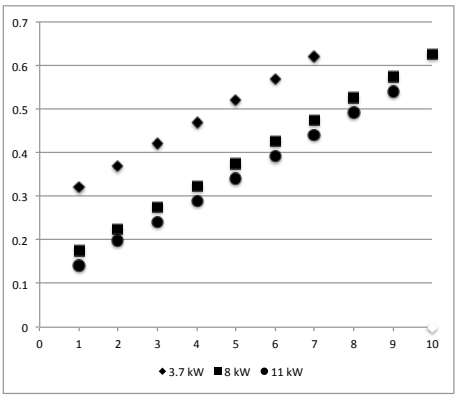


Fig. 1 Profit values for the instances of an activity a_i and $\alpha = 0.5$. The X axis depicts the charged amount relative to the maximum demand $d_i/dmax_{a_i}$

The MIP formulation becomes
Problem EVRSTW:

$$Z = \max \sum_{i \in I} p_i x_i \tag{2}$$

$$s.t. \sum_{i \in I | a_i = j} x_i = 1, j \in A \tag{3}$$

$$\sum_{i \in I | t \in [s_i, f_i]} w_i x_i \leq P^t, t \in T \tag{4}$$

$$\sum_{i \in I | m_i = m} v_i^t x_i \leq 1, m \in M, t \in T \tag{5}$$

$$x_i \in \{0, 1\}, i \in I \tag{6}$$

The objective in (2) is to maximize the total profit. The equality (3) makes sure that exactly one instance of each activity is selected. The inequality (4) limits for each time slot t the total power due to contributed instances. Finally, the contraints (5) avoid that two cars park ($v_i = 1$) on the same parking place m in the same time slot.

3 Solution approaches

Three different heuristics have been implemented and tested: greedy, local ratio and min-conflicts. For reference, exact solutions have been obtained using an AMPL formulation and the CPLEX solver.

3.1 Simple greedy heuristic

The greedy heuristic takes only those instances generated by usage of minimum charging speed (w_{min}) which start at the beginning od the activity's time window (e_j). The steps are shown in Algorithm 1.

Algorithm 1 Greedy Heuristic scheduling

```
for all a ∈ A do                                   ▷ A - activities
    I ← IG.getGreedyInstancesFor(a)
    if I is empty then
        continue
    end if
    I.sort()                                       ▷ according to profit
    for all i ∈ I do
        if i is feasible then                      ▷ doesn't violate the power constraint
            a.setInstance(i)
        end if
    end for
end for
```

As expected, this approach performs poorly. The first issue is the activity's time window - if it is too short, the list generated by the instance generator will be empty. This happens when even the minimum energy demand ($dmin_j$) of the activity can not be satisfied by the charging speed of w_{min} within the parking duration. The second issue is the usual problem of the greedy algorithms: if the current instance can not fit into the schedule, it gets thrown away and never observed afterwards. These two factors cause that many activities are not being scheduled at all.

3.2 Local ratio heuristic

The local ratio algorithm belongs to a class of stack based heuristics [7], and has been proposed in packing problems to maximize bandwidth throughput [8], [5]. In our case the bandwidth corresponds to the charging power. For this type of problems, the algorithm can achieve a 1/3 approximation, which gets better as the ratio between the charging power (speed) of individual activities and the total available charging power is smaller. The algorithm works as follows: the instances I are first sorted in increasing order of their end (charging) times. In the first sweep we select an instance with minimum end time and decrease the profits of all instances that a) belong to the same activity, and b) overlap with the selected instance. Instances with non-positive profits are removed, and the selected activity is pushed on a stack. When no activities remain unconsidered, in the second sweep activities are popped from the stack and those who violate the total power constraint (4) are deleted. Denote the selected instance $\tilde{i}$, then the profits of conflicting instances p_i are reduced as follows: $p_i = p_i - \beta \bar{w_i} p_{\tilde{i}}$, where $\beta = 2$. Other than in the exact algorithm, the amount of resource has to satisfy $0 \leq \bar{w_i} \leq 1$. The problem arises at the calculation of if $\bar{w_i}$ as w_i varies in the interval $[s_i, f_i]$. The conservative rule in (7) would only waste resources because even a short drop in the capacity (available power) would affect all the charging periods around it.

$$\bar{w_i} = \min_{t \in [s_i, f_i]} w_i \tag{7}$$

The algorithm and its implementation are described in [5].

After solving the integer problem exactly (which turned to be excessively time-demanding, even non-feasible for bigger problem instances, due to the problem's NP-hard nature), followed by the use of greedy and local ratio heuristics (which had the problem of not scheduling all the activities), we investigated the application of min-conflicts heuristic.

3.3 Min-Conflicts Heuristic

This section describes a new solution for our problem based on the main ideas of min-conflicts heuristic. This method was proposed by Minton et al. [14] and it has been proven very useful on solving different constraint satisfaction problems. Min-conflicts heuristic has been applied, among others, for the Hubble Space Telescope scheduling problem [14], Workforce Scheduling [18], Break scheduling [17], N-Queens problem [14], etc. This technique has also been used in combination with noise strategies such as random walk [15]. Furthermore, a similar algorithm to Min-Conflicts heuristic is currently one of best local search strategies for solving Boolean satisfiability problems [16].

Instead of constructing feasible solution gradually, the min-conflicts heuristic generates suboptimal initial solution and then improves it iteratively. The main idea of this method is to concentrate the search in the neighborhood of variables that are in conflict. This technique initially assigns random values to all variables of the problem to be solved. The current solution is then iteratively improved with the following strategy: in each iteration min-conflicts selects randomly a variable that is in conflict (i.e., the variable appearing in a constraint that is still violated). The new value that minimizes the number of conflicts is assigned to the selected variable. The ties are broken

randomly. Note, that different strategies can be applied to escape from the local optimum. If the number of conflicts can not be minimized, even the solutions with more conflicts or same number of conflicts can be accepted. Alternatively, random strategies like random noise or random walk can be introduced in the min-conflicts method.

In order to apply the min-conflicts strategy for our problem, we first formulate it as a constraint satisfaction problem consisting of variables, their domain and constraints. In our case the activities (cars) represent the variables, the instances (charging plans) represent the values, and any timeslot-wise power constraint violation represents a conflict. The final state with no conflicts is when the power consumption in each timeslot is **smaller than or equal to** the available power, i.e. when the power constraint is not violated.

3.3.1 Internal Solution Representation

The Schedule **S** is internally represented as a two-dimensional MxT matrix. Each timeslot represents a 15 minute interval. In this matrix, the time window of an activity is marked (in the row corresponding to the machine m) with the activity ID. A positive activity ID value denotes, that the vehicle is charging, whereas a negative value means "parking only". An example of a schedule is shown in Figure 2.1. This data structure is significant for the performance of the algorithm, since it allows for rapid access to the activities while traversing the search space. A hashed map is therefore used, with activity IDs as keys. The map leads to the information about the activities and the assigned instances (charging plans).

```
::: Schedule :::
  0   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 |  m
-------------------------------------------------------------------------------------------------------------------------------------
  0   0   0   5   5  -5  -5  -5  -5  -5   3   3   3   3   3 -17 -17  17  17  17   0   0   0   0   0   0   0   0   0   0   0 |  0
  0   0   0   0   4   4   4   4   4   4  -4  -4 -10 -10 -10 -10 -10 -10 -10 -10  10  10  10   0   0   0   0   0   0   0   0 |  1
  0   0   0   0  -7   7   7   7   7   7 -11 -11  11  11  11  11  11  11  11  -8  -8  -8  -8   8   8   8   8   8   0   0   0 |  2
  0   0   0   0 -12 -12 -12  12  12  12  12  12  15  15  15  15  15 -15 -15  20  20  20  20  20   0   0   0   0   0   0   0 |  3
  0   0   0   0   0   1   1   1   1  -1  13  13  13  13  13  13  13 -13 -13   0   0   0   0   0   0   0   0   0   0   0   0 |  4
  0   0   0   0   0  19  19 -19 -19 -19 -19 -19 -19 -19   0   0  -9  -9  -9  -9   9   9   9  -9  -9   0   0   0   0   0   0 |  5
  0   0   0   0   0   0   0   0   0 -14  14  14  14  14  14  14  14 -14  -6  -6  -6  -6  -6   6   6   6   6   0   0   0   0 |  6
  0   0   0   0   0   0   0   0   0   2   2   2   2   2  -2  -2   0   0 -18 -18 -18 -18 -18 -18  18  18  18  18   0   0   0 |  7
```

Fig. 2 Example schedule internal representation

3.3.2 Initial Solution

The generation of an initial solution is done in several different ways. Initial instances (charging plans) have been assigned to activities by one of the following rules:

a) *random*: assign a random instance
b) *greedy max-profit-first*: assign a maximum profit instance, but only among the ones with a low charging speed of 3.7 kW (*greedy*); if such does not exist, turn to a)
c) *max-profit-first*: assign a maximum profit instance among all instances

The technique used depends on the problem size and on the mode of operation of the scheduler: online or offline. For the offline mode, when problem instances are bigger (e.g. more than 40 activities), *max-profit-first* technique creates an infeasible initial solution. In the online mode, when the activities are added into the schedule one by one at different points in time, this method performs well.

3.3.3 Optimization

After the initial instance has been assigned to each of the activities, the optimization process starts. The method is presented in Algorithm 2.

Algorithm 2 Optimize

```
negativeSlots ← getNegativeTimeslots()
while negativeSlots not empty and timeLimit not reached do
    timeslot ← a random timeslot from negativeSlots
    activity ← a random activity charging at timeslot
    newInstance ← bestImprovement(activity)
    activity.setInstance(newInstance)
    updateSchedule(activity)
    negativeSlots ← getNegativeTimeslots()
end while
```

`negativeSlots` is a list of timeslots in which the difference between available power and power consumption is negative. The optimization stops when there are either no more timeslots with negative power differences or when the time limit is reached. In the `bestImprovement` procedure, the solution is iteratively improved. Its framework is shown in Algorithm 3.

Algorithm 3 Best Improvement

```
bestInstance ← activity.getInstance()
bestPDiff ← calculatePowerDifferences()
instances ← IG.getInstancesFor(activity)
if p ≤ 0.1 of a random number [0,1] then
    return a random instance from instances
end if
sort instances by profit*
for i ∈ instances do
    newPDiff ← updatePowerDiff(i)
    if sumNegatives(newPDiff) > sumNegatives(bestPDiff) then
        bestInstance ← i
        bestPDiff ← newPDiff
        if sumNegatives(newPDiff) == 0 then
            break
        end if
    end if
end for
return bestInstance
```

*The sorting step, depends again on the system mode and the size of the problem instance. In the offline mode it is performed only in case of $|activities| <= 20$, due to feasibility concerns.

The `calculatePowerDifferences` function calculates the difference between the available power and the power consumption in each timeslot. This is also used in the function `getNegativeTimeslots` in Algorithm 1.

The `updatePowerDiff` function is a quick way of determining improvement that the given instance i_{new} offers. It iterates through the timeslots, deduces the power used by the i_{old}, and adds up the power of the newly assigned instance i_{new} in their

"active" timeslots. If the sum of the negative values in `newPDiff` array is greater (i.e. less negative) than the previous best, then i_{new} is assigned as the new `bestInstance`.

The `sumNegatives` function serves as a fitness function here, and it sums up the negative values in the power differences array. This is a fast method of determining the local fitness improvement. It is possible to do so, since all the instances of other activities remain fixed.

4 Computational Results

The described heuristics for the offline scheduling problem EVRSTW have been implemented and solved a set of synthetic generated problem sets. The problems were solved in an *offline mode*, e.g. all the activities were fed into the algorithm at once. The schedule is then built "from scratch", and the algorithm works with the whole problem.

The problems were selected to simulate a moderate load, meaning that the solution contains several EVs that cannot charged with the maximum requested energy. The problems are characterized by [A]-[M]-[P]: the number of activities (A), charging points (M) and power limit P, which is kept constant during the time horizon T. All the problems are feasible, i.e. there are enough machines, and each charging activity can get the minimum requested energy amount. Along with the list of activities, for each problem instance the number of machines and a power limit are given. The former is an integer value and depicts how many charging spots we have at hand, while the latter is either a single real value or an array of values,e.g. the profile of the available power which varies throughout the day.

The problems were generated using the uniform distribution U(a,b): e_j= U(0,20), l_j= e_j+ U(4,10), $dmin_j$= U(3,5), $dmax_j$= $dmin_j$+U(3,5).

Config.	GH	LRH	MCH (min-avg-max)	CPLEX
20-16-30	10	15.16	14.32 15.59 17.15	19.20
40-32-40	11.47	23.68	22.30 21.59 23.59	29.60
60-40-55	12.96	34.16	32.11 32.68 33.53	41.48
80-48-70	17.28	45.78	41.85 43.34 44.74	53.47
100-56-90	22.52	59.81	53.91 55.15 56.38	67.93
120-64-100	28.76	69.09	64.91 66.34 73.56	78.52

Table 2 Objective function for different algorithms, $\alpha = .95$

Config.	GH	LRH	MCH (min-avg-max)	CPLEX
20-16-30	.006	0.16	1.35	24
40-32-40	.066	0.34	0.14	26
60-40-55	.078	0.36	1.10	192
80-48-70	.186	0.46	5.98	527
100-56-90	.202	0.51	8.88	1500+
120- 64-100	.155	0.73	111.70	468

Table 3 Runtime of algorithms (s) (average over 50 runs)

Config.	GH	LRH	MCH (50 run avg.)
20-16-30	1	2	0
40-32-40	14	11	0
60-40-55	22	18	0
80-48-70	30	25	0
100-56-90	35	30	0
120- 64-100	45	39	1

Table 4 Refused charging activities (unscheduled)

We compared the heuristic results with an exact solution from a AMPL/CPLEX mixed integer model, setting the stop condition at an optimality gap of 1%. The test results are showed in the Tables 2, where the objective function was computed according to definition (1), Table 3 in which the runtime is seconds is given for all the algorithms, and Table 4 in which certain activities are removed in order to achieve feasibility of GH and LRH. The use of CPLEX has the only disadvantage that, for harder or larger problem instances, it will not terminate with a slightly infeasible solution in due time, although such a solution would be acceptable in the environment.

We observe that MCH performs better than the two other heuristics, achieving up to 80% of the optimal solution. The execution times are moderate, one single instance of the largest problem remained infeasible after the time limit of 300s. The main advantage of MCH upon greedy and local ratio heuristics is the feasibility, expressed here by the number of refused (unscheduled) activities. The objective (fitness) is not significantly better, which is due to the fact that the min-conflicts algorithm is a method to solve constraint satisfaction problems (CSP), and not constraint optimization problems (COP). Therefore, there is definitely room for further improvement in the sense of optimization. The heuristic presented here stops immediately after it reaches feasibility, i.e. the moment it finds the schedule which does not violate the power constraint in any of the timeslots. Fitness calculation could be further modified towards prefering better solutions among the feasible ones.

Another aspect that created less problems than were expected was the memory requirement for the large number of instances generated. This number depends on combination of variables, especially the size of the time windows.

5 Real world scheduling

The scheduling algorithm using MCH was integrated in a controller that can react to events generated from cars and from the power grid distribution system. The main differences to the static setting are:

1. the schedule is rolling a timeslot at a time, and the system is now able to process different events such as reservations, arrivals within or outside of the planned period, leave and timeout events. The scheduler checks of course whether a machine is still busy, when new activities are allocated.
2. at any time the scheduler deals with a subset of the activities (only those within the time horizon), reducing the number of problem instances. Additionally, the same activity can be rescheduled as long as the selected instance has not started yet. This allows a continuous improvement of the schedule.
3. the sum-power constraint in Eq. (4) is the result of the available power prediction and may take different values during the duration of an activity (instance) as shown

in Fig. 3. This subtile constraint is believed to make the online problem more difficult to solve. A good schedule algorithm would allocate the activities tight around the power drop at 4 p.m. as shown in Fig. 3.

4. the online problem has to deal with prediction errors.

Typical prediction errors occur at the estimation of the leave time of a vehicle: early leaving leads eventually to interruption of charging, but also leaving later than planned leads to a suboptimal schedule. In the following experiments we compare the quality of the online solution to that of an offline optimal decision. Therefore, we first run the online scheduler for one day simulation, using a realistic vehicle charging load distribution, see Fig. 3. The actual parking duration with an average of 3.5 hours is statistically modified to simulate prediction errors, according to a normal distribution. The generated activities with their actual leave times are scheduled by an offline scheduler for comparison. The experiment should show, how much better is the offline scheduler for different levels of the prediction error (which we define by the standard deviation) in the real versus planned duration.

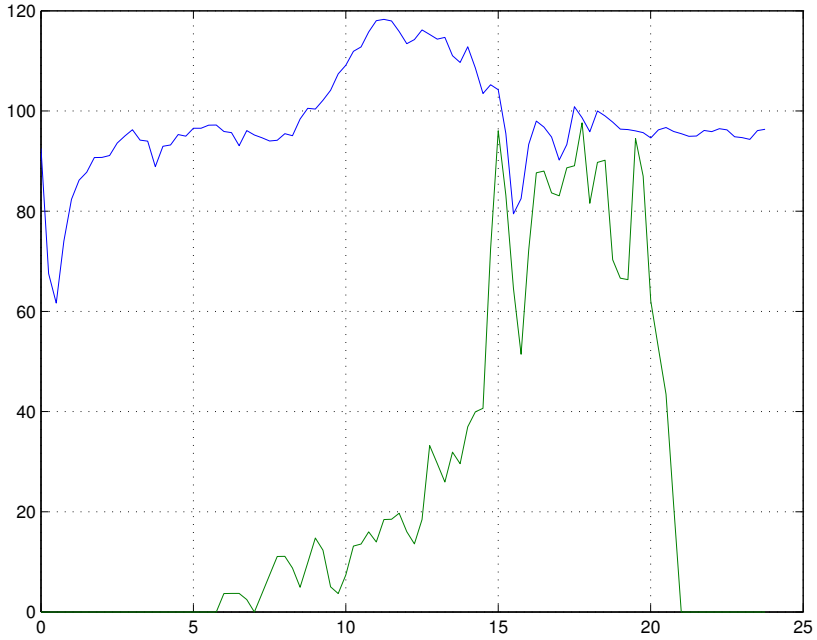


Fig. 3 Charging profile at a shopping mall: Total power limit P_t and total load as a function of time of the day (hour).

The results in Table 5 show as expected that the performance of the scheduler in online mode looses another 9% when the average planned duration of 3.5 hours varies with a standard deviation of half an hour. This result is of course specific to the "shopping mall" car arrival distribution as illustrated by the load curve in Figure 3.

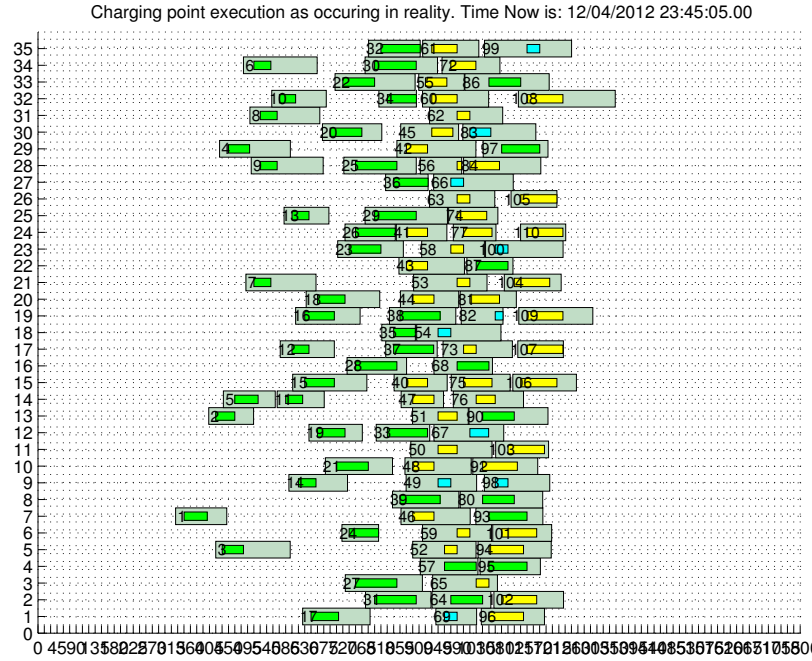


Fig. 4 Online schedule at the end of the day, each tick on the x axis is 15 minutes, the machines 1-35 are on the y axis; the charging speeds are 3.7 (green), 8kW (yellow), 11kW (turquois), the gray rectangles are the time windows.

duration error σ [minutes]	objective ratio online/offline [%]	total charged ratio online/offline [%]
0	88%	83%
30	79%	75%
60	76%	72%

Table 5 Online Scheduling performance compared to offline optimum, under parking duration prediction error (80 cars)

6 Conclusions and Future Work

In this work we analyzed a scheduling problem that occurs when several electric vehicles are recharged at a public charging station. The specific application requirements (time windows, discrete interval, minimum and maximum energy demand) led to the integer program formulation of a resource allocation problem. We showed that a min-conflicts heuristics produces promising results and achieves during a limited runtime (of 10 seconds) and in practical cases around 80% of the optimum obtained by solving exactly the integer program. The heuristics has been integrated in an online environment. The uncertainity in the parking duration has a significant impact on the solution quality, especially in the high load case, where the charging can only be scheduled towards the end of the parking period.

Future work will explore multi-objective functions that reflect different stakeholder interests, such as equipment costs or time varying energy costs. Furthermore, ways to make the schedule more robust to metering data errors in general will be investigated.

Acknowledgements This work has been conducted within the Austrian project KOFLA, which is funded by the programme "ways2go" and the Austrian Federal Ministry for Transport, Innovation and Technology. Nysret Musliu was supported by the Austrian Science Fund (FWF): P24814-N23.

References

1. J. A. Pecas Lopes, F. J. Soares, P. M. Almeida, M. Moreira da Silva, Smart Charging Strategies for Electric Vehicles: Enhancing Grid Performance and Maximizing the Use of Variable Renewable Energy Resources, EVS-24, Stavanger, Norway, May 13-16, 2009.
2. K. Clement, E. Haesen, and J. Driesen, Coordinated Charging of Multiple Plug-In Hybrid Electric Vehicles in Residential Distribution Grids, PES Power Systems Conference and Exposition, 2009.
3. O. Sundstrom and C. Binding, Planning Electric-Drive Vehicle Charging under Constrained Grid Conditions, International Conference on Power System Technology, 2010.
4. S. Sojoudi and S. H. Low, Optimal charging of plug-in hybrid electric vehicles in smart grids, IEEE Power and Energy Society (PES) General Meeting, Jul. 2011.
5. Bar-Noy, Amotz, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber, A unified approach to approximating resource allocation and scheduling, Journal of the ACM (JACM) 48, no. 5 (2001): 1069-1090.
6. Bo Chen, R. Hassim, M. Tzsur, Allocation of bandwidth and storage, IIE Transactions, Vol. 34, Iss. 5, 2002.
7. Allan Borodin, David Cashman, and Avner Magen, How well can primal-dual and local-ratio algorithms perform?, ACM Trans. Algorithms 7, 3, Article 29 (July 2011).
8. Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz, Local ratio: A unified framework for approximation algorithms, in Memoriam: Shimon Even 1935-2004, ACM Comput. Surv. 36, 4 (December 2004), pg. 422-463.
9. Sandford Bessler, Jesper Grønbæk, Max Herry, Andreas Schuster, Rupert Tomschy, Supporting E-mobility Users and Utilities towards an Optimized Charging Service, European Electric Vehicle Congress, Brussels, 26-28 october 2011.
10. Andreas Darmann, Ulrich Pferschy, Joachim Schauer, Resource allocation with time intervals, Theoretical Computer Science, Volume 411, Issue 49, Pages 4217-4234, 5 November 2010.
11. G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani, Improved approximation algorithms for resource allocation, in 9th International Integer Programming and Combinatorial Optimization Conference, volume 2337 of LNCS, pages 401414, Springer, 2002.
12. Phillips, Cynthia A., R. N. Uma, and Joel Wein, Offline admission control for general scheduling problems, in Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, pp. 879-888, Society for Industrial and Applied Mathematics, 2000.
13. B. Chen, R. Hassin, and M. Tzur, Allocation of bandwidth and storage. IIE Transactions, 34:501507, 2002.
14. Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird, Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence, 58:161–205, 1992.
15. Richard J Wallace and Eugene C Freuder, Heuristic methods for over-constrained constraint satisfaction problems, in *Overconstrained Systems, Springer 1106*, 1996.
16. Bart Selman and Henry A. Kautz and Bram Cohen, Local Search Strategies for Satisfiability Testing, Proceedings of the Second DIMACS Challange on Cliques, Coloring, and Satisfiability, 1993.
17. Andreas Beer, Johannes Gärtner, Nysret Musliu, Werner Schafhauser, Wolfgang Slany, An AI-based break-scheduling system for supervisory personnel, IEEE Intelligent Systems, 25(2):60-73, 2010.
18. Nysret Musliu, Heuristic Methods for Automatic Rotating Workforce Scheduling, International Journal of Computational Intelligence Research, Volume 2, Issue 4, pp. 309-326, 2006.

MISTA 2013

Directing selection within an extended great deluge optimization algorithm

Ryan Hamilton-Bryce • Paul McMullan • Barry McCollum

Abstract

The original Extended Great Deluge algorithm uses a stochastic method for selecting which examination to use in the neighborhood search process. This paper a variation on the Extended Great Deluge optimization algorithm, by directing examination selection within an optimization heuristic, utilizing a weighted list gathered during adaptive construction to influence examination selection. Two approaches to utilizing the weighted list of examinations are studied; utilizing the list at the end of the construction phase and evolving the list over the duration of the improvement phase. The approach described in this paper is able to produce feasible solutions for all of the publicly available datasets from the 2007 International Timetabling Competition, within a time limit specified by the competition organizers. In addition, the results generated with the new selection technique compare favorably with those of the original Extended Great Deluge approach. The results are also competitive with those of the competition winner and with research conducted post competition which observe the time limits specified within the competition rules.

1 Introduction

The challenge of producing solutions for timetable problems such as Course and Examination timetabling involves a combination of practical and research based approaches [1]. Due to the complexity of the problem and the time required to solve these problems using discrete methods, research has focused on the use of search based heuristic techniques. Within the area of examination timetabling, progress in research has been facilitated by the availability of benchmark data sets[2,3]. Results using a wide range of techniques have been reported, with varied levels of success based on generality and time taken [4]. A successful technique can be viewed as one which can produce feasible and workable solutions to a range of differing problems within a problem domain, in a practical timescale.

The 2nd International Timetabling Competition (ITC2007) [2] introduced an examination scheduling track, based on the post-enrollment examination timetabling problem. This track introduced a number of new datasets, drawn from anonymised data from several institutions

worldwide. New result sets continue to be validated using the competition's online validation service despite the competition closing almost five years ago. The next timetabling Competition (ITC2013) to be announced will further develop the problem definition, to further extend the real world aspects of research in this area and to encourage innovation with the development of new problem solvers.

The Extended Great Deluge (EGD) [5] which had previously been very successfully introduced to Course scheduling, was adapted and used for the ITC 2007 Examination data sets [6]. This technique was able to produce feasible and competitive solutions for all of the data sets presented in the ITC2007. It was reported that a link exists between the initial adaptive construction phase and the stochastic Extended Great Deluge optimization phase [7], which could be explored for enabling possible improvements on the quality of timetables generated as part of the timetabling process without increasing the length of time required.

The EGD uses two phases for scheduling; construction, to create an initial feasible solution and; improvement, to explore the solution space of the constructed solution for further better solutions. During adaptive based examination timetable construction, an ordered list of “difficult” (hard to schedule) examinations, is maintained [8]. When the improvement phase is reached, this information is discarded in favor of a fully stochastic selection routine, to allow as much time as possible in trying to find an optimal solution within the neighborhood search space. This technique was able to produce results that beat those of Müller, the ITC2007 winner in six of the twelve datasets when run on identical hardware [6].

In the work presented here, co-operation between an adaptive construction phase and a great deluge based optimization phase is studied. Information gathered and used during the construction phase is used to direct examination selection within the improvement phase. Two selection criteria have been identified using the information passed from the adaptive construction phase to direct examination selection within the improvement phase. The first selection criteria uses the list provided by the adaptive scheduling technique, as it exists in its final state at the end of the construction phase. The second criteria also initially uses the list in this form, however with each iteration of the optimizer this list order is updated with new weightings, based on penalties as a result of soft constraint violations introduced. The second criterion allows the list to evolve across the entire duration of the optimization process. For each of the criteria, the effect of supplementing the existing stochastic selection method with information gathered during adaptive construction and replacing the stochastic selection method entirely with an iterative method utilizing the weighted list is studied. An attempt will also be made to identify any trends within the adaptive list, to provide the greatest benefit to the optimization process.

The remainder of the paper is as follows: Section 2 briefly describes the examination timetabling problem. Section 3 describes Squeaky Wheel construction and the Extended Great Deluge optimization algorithm. Section 4 describes how examination selection was directed for the optimization process using information gathered during adaptive construction. Section 5 describes the experimental environment and time parameters used during experimentation. Section 6 presents and discusses the results and finally section 7 concludes the paper with a brief discussion on the effectiveness of the technique and potential future research areas.

2 The Examination Timetabling Problem

Examination timetabling, as a subset of the general timetabling problem, has been proven to be NP-hard [9]. Examination timetabling is an attempt at trying to allocate a set of events (exams), into a series of available resources (timeslots and rooms), subject to a series of constraints. Primarily, there are two types of constraints; hard and soft. Hard constraints must be satisfied for a timetable to be considered feasible, for example an exam cannot be scheduled in a room that is too small for the capacity, or a student must not have two exams at the same time. Soft constraints on the other hand represent desirable preferences, which are not required to be satisfied for the timetable to be considered feasible, but may affect the fitness or ”quality” of the solution desired by the institution. For example, while it may not be preferable for a student to have two exams in one day, a timetable can still be considered feasible if this

does occur. The main goal when solving this problem is to minimize the number of soft constraint violations, while at the same time maintaining a feasible solution. As it is possible to assign a numeric value to the quality of a timetable based on how well it satisfies the various constraints, it is possible to directly compare two timetables, where the timetable with the lower overall penalty is considered the more acceptable solution.

Examination timetabling can be considered as either a pre-enrolment or post-enrolment problem. In pre-enrolment, the timetable is constructed prior to students completing their enrolment process, with timetables being based on either predicted class size, or an absolute maximum class size, as well as detailed knowledge of the required curriculum structure defined by the institution. In post-enrolment, the student enrolment data is known at the time of scheduling, allowing a potentially more accurate use of the available resources during the examination period. It is also possible for an initial pre-enrolment timetable to be produced and then optimized with post-enrolment data once student enrolment has completed.

The requirements for real-world examination timetabling problems are often unique for each individual institution, with the type and mix of hard and soft constraint options reflecting the preferences of the institution in question. However it is possible to identify a common set of both hard and soft constraints for benchmark and research use.

Carter, et al. introduced a set of 13 benchmark examination datasets in 1996 [3], drawn from three Canadian high schools, five Canadian universities, one American university, one British university and one university in Saudi Arabia. These datasets have been widely tested and used in examination timetabling research [4]. These datasets were supplemented by a series of new datasets, drawn from anonymised data provided by several institutions worldwide, for the 2007 International Timetabling Competition (ITC2007).

	Exams	Students	Periods	Rooms	Conflict Density	Period Hard Constraints	Room Hard Constraints
Exam 1	607	7891	54	7	5.05%	12	0
Exam 2	870	12743	40	49	1.17%	12	2
Exam 3	934	16439	36	48	2.62%	170	15
Exam 4	273	5045	21	1	15.00%	40	0
Exam 5	1018	9253	42	3	0.87%	27	0
Exam 6	242	7909	16	8	6.16%	23	0
Exam 7	1096	14676	80	15	1.93%	28	0
Exam 8	598	7718	80	8	4.55%	20	1
Exam 9	169	655	25	3	7.84%	10	0
Exam 10	214	1577	32	48	4.97%	58	0
Exam 11	934	16439	26	40	2.62%	170	15
Exam 12	78	1653	12	50	18.45%	9	7

Table 1 – ITC 2007 Dataset Information

Table 1 lists the main characteristics for each of the examination datasets provided by the organizers of ITC2007. The conflict density is a measure of the number of examinations that are in conflict due to student enrolment, defining how tightly the problem is constrained by student module choice. It is initially observed that the conflict density for most of the datasets is quite low, which is reflective of the amount of choice available to students within a modern curriculum, with a large variation in course or subject choices between each student. The measure of problem “size”, based on the number of exams and students, varies across the datasets. The largest exam dataset could be argued to be either Exam 3/Exam 11 or Exam 7 and the smallest to be either Exam 9 or Exam 12. The amount of periods and rooms available will also have a measurable effect on the difficulty of constructing a feasible solution. Exam 3 and Exam 11 are almost identical, however Exam 11 has a much smaller set of period resources available. The differences between Exam 3 and Exam 11 reflect a "real-world" situation where an examination session has been shortened to minimize space and staff costs, while keeping all other existing constraints where possible.

Recent attempts to solve the examination timetabling problem continue to involve a variety of different techniques. Ant Algorithms, introduced by Dorigo [10] and implemented for the examination timetabling problem by Eley [11], were among the first parallel implementations to solve the problem. Each ant works concurrently and independently to build

a complete, or partial solution starting from an initial state defined by problem dependent criteria. Genetic Algorithms [12] are modeled on Darwin's theory of evolution. Once an initial population has been constructed, it is refined over a series of iterations, with a fitness function calculating the fitness of each individual within the population. Late Acceptance Hyper-heuristics were introduced by Burke and Bykov [13]. Traditionally, the approach in hyper-heuristics was to compare the current solution with the solution immediately preceding. In late acceptance, the current solution is compared with what was the "current" solution a number of iterations previously. Late acceptance techniques are able to produce competitive results in a short timeframe. Reinforcement Learning [14] techniques are used to influence heuristic selection for hyper-heuristics. A memory log of heuristic actions is kept during execution, with successful actions being rewarded and unsuccessful actions being punished. With this log, successful actions are chosen more often and unsuccessful actions are chosen less often across the search space. Both long term [14] and short term [15] memories have been explored in this technique. Tabu Search [16],[17] is a local search based technique. Unlike other such techniques, it maintains a list of solutions that have recently been visited, which is used to prevent the optimizer from repeatedly considering similar neighborhoods, helping to avoid local optima.

3 The Extended Great Deluge Algorithm and Squeaky Wheel Construction

The Extended Great Deluge (EGD) algorithm was introduced in 2007 by McMullan [5] as an improvement on the Great Deluge (GD) algorithm introduced by Dueck [18], itself an improvement on Simulated Annealing. Generally, a GD algorithm process will self-terminate when a lack of improvement has been observed for a specified amount of time or iterations, as it is unlikely that further significant improvement will be observed within a practical timescale. Instead of terminating, the EGD employs a reheat function to widen the boundary level, as well as adjusting the cooling parameter. This allows the EGD to exit potential local optimum states.

The EGD algorithm was introduced to solve the University Course timetabling problem [5] and was later applied to the examination timetabling problem [6]. In both approaches, the authors combined an EGD optimizer with a Squeaky Wheel (Adaptive) constructor. During experimentation, it was found that it was preferable for construction to continue, for approximately 9% of the entire execution time of the process, providing a reasonable timetable to the optimization phase.

```
Set the initial solution s using a construction heuristic;
        Calculate initial cost function f(s)
        Set Initial Boundary Level B0 = f(s)
        Set initial decay Rate •B based on Cooling Parameter
        While stopping criteria not met do
                Apply neighbourhood Heuristic S* on S
                Calculate f(s*)
                If f(s*) <= f(s) or (f(s*) <= B Then
                        Accept s = s*
                Lower Boundary B = B – •B
        If no improvement in given time T Then
                Reset Boundary Level B0 = f(s)
                Set new decay rate •B based on Secondary Cooling Parameter
```

Psuedo-code 1 - Extended Great Deluge

Squeaky Wheel (adaptive) construction [19] begins with an iterative construction process, building an initial schedule by placing one exam at a time, in the order determined by a weighted sequence. Each exam is assigned to the first available time and room combination, where possible, ensuring that a feasible solution is maintained while minimizing soft constraint violations. If an exam cannot be scheduled in the current iteration, it is left unscheduled and the constructor moves onto the next exam. When an exam is scheduled a weighting based upon its current penalty, as defined by the various soft constraint violations, is added to the stored

weighting in the weighted list. If an exam cannot be scheduled a suitably large weighting is used instead. Once an attempt has been made to schedule all exams, the weighted list is re-sorted and subsequently those exams with the highest weighted value are first to be scheduled on the next iteration or "run" of the constructor. The weightings held in the list evolve over the duration of the entire construction process.

```
Read in the problem file into memory and build conflict and suitability matrices
Calculate an initial weighting based on pre-defined criteria
While stopping criteria not met do
        For each exam(ei) in weightedList do
                For all suitable timeslots(k) of ei do
                        Get ti from k
                        If canSchedule (ei, ti)
                                Calculate best penalty and store best (bestti)
                If multipleBest found then
                        Schedule(ei, randomBestti) and store associated weighting in weightedList
                Else If bestti found then
                        Schedule (ei, bestti) and store associated weighting in weightedList
                Else
                        Leave exam unscheduled and add large weighting in weightedList
        Sort(weightedList)
```

Pseudo-code 2 - Squeaky-wheel (Adaptive) construction

4 Directing examination selection

Traditionally, the weighted list of examinations built during Squeaky Wheel construction is discarded once the process moves from construction to improvement or optimization. However, in [7] a link was theorized between a Squeaky Wheel based construction phase and the Extended Great Deluge based optimization phase. It is believed that the information that is traditionally discarded can be fed into the optimization phased and used to direct the selection of examinations for the neighborhood search process. The directionRatio is a new variable introduced with this technique, which defines how often the weighted list is used to direct exam selection. The initial experimentation into this approach focused on utilizing the weighted exam list as it existed at the end of Squeaky Wheel construction.

```
Build an initial solution using an adaptive construction heuristic
Calculate initial penalty f(s)
Set initial boundary level B0 = f(s)
Set initial decay rate DR based on cooling parameter
While stopping criteria not met do
        Select optimization heuristic //move or swap
        If directionRatio met
                Select exam from weightedList
        Else
                Select random exam
        Apply optimization heuristic on exam
        Calculate penalty of new timetable f(s*)
        If (f(s*) <= f(s)) or (f(s*) <= B) Then
                Accept s = s*
        Lower boundary B = B - DR
        If no improvement in time T then
                Reset boundary level B0 = f(s)
                Set new decay rate DR based on next cooling parameter
```

Pseudo-code 3 - Directed Extended Great Deluge (DEGD)

Once this experimentation was complete, it was decided to examine the effect of updating and re-ordering the weighted list, based on the same criteria used during Adaptive construction. It was initially thought that it would be possible to update the order of the list after each successful optimization, however sorting the list proved to be too processor intensive for the larger datasets, reducing the number of optimization iterations significantly and having a negative effect on the overall quality of the final timetable. A suitable

compromise was found; the weighted list would continue to update upon a successful optimization and re-ordering the weighted list would occur during a reheat action. This compromise allows the weighted list to continue to evolve across the duration of the optimization process with a minimum of computational expense.

5 Experimental Environment

The algorithm was implemented and tested on a PC with an Intel Core i5-2400 3.1GHz processor, 8GB RAM and Windows 7. The program was coded in C# targeting the .NET Framework 4.5. For each problem set, the program was executed for ten iterations, with a 235 second time limit per iteration determined by a benchmarking application released by the competition organizers. During initial experimentation it was found that allowing adaptive construction to continue for approximately 30% of the total execution time provided the best results with the new code.

6 Results and analysis

The results below are generated based upon an initial random seed, which has been recorded to ensure repeatability of the experiments. The random seeds can be produced upon request. Where a method has been highlighted in bold and underlined, it is considered to be the best method for solving the given problem set. An individual solution is considered better when both the average penalty and the consistency of the generated results are lower than those of its comparison.

Initial experimentation identified that splitting the weighted list into thirds provided a greater improvement than any other split. There are nine new selection techniques under investigation:

i) First Third - randomly select exam from first third of weighted list
ii) Second Third - randomly select exam from second third of weighted list
iii) Third Third - randomly select exam from final third of weighted list
iv) Variable - randomly select exam from a portion of the weighted list, the portion of the list used is dictated by a set of ratio parameters, for example the first third of the list is used for five in every eight iterations, the second third is used for two in every eight iterations and the final third is used for one in every eight iterations
v) Full List – randomly select an exam from anywhere in the weighted list
vi) Sequential - iteratively process through the weighted list, starting with the hardest to schedule examination
vii) Sequential Re-sort – iteratively process through the weighted list, starting with the hardest to schedule exam. The weightings are updated upon each successful optimization and re-sorted on every reheat
viii) Sequential Reverse - iteratively process through the weighted list, starting with the easiest to schedule examination
ix) Sequential Reverse Re-sort - iteratively process through the weighted list, starting with the easiest to schedule exam. The weightings are updated upon each successful optimization and re-sorted on every reheat

The new selection techniques have been applied to two combinations of neighborhood search heuristics:

1. Move Only – the selection technique is only used for selecting an examination for moving, swap continues to use a completely stochastic method
2. Move and Swap – the selection technique is use for selecting an examination for both moving and swapping. For swapping, the selection technique only selects the first examination, the other examination is picked randomly

		Exam 1				
		Best	Worst	Average	Range	Standard Dev.
	Original EGD	5377	5865	5598.5	488	198.9630
Move Only	First Third	5350	5866	5622.1	516	150.6176
	Second Third	5310	5877	5620	567	168.9819
	Third Third	5468	5874	5618.3	406	137.5436
	Variable	5385	5878	5615.7	493	184.4180
	Full List	5468	5874	5621.9	406	141.9636
	Sequential	5203	5798	5515.6	595	186.9909
	Sequential Re-sort	5236	5680	5474.4	444	139.2673
	Sequential Reverse	5454	5978	5704.9	524	161.1945
	Sequential Reverse Re-sort	5312	6106	5738.2	794	235.7158

Table 2a - Full results for Exam 1 for the Move Only heuristic

		Exam 1				
		Best	Worst	Average	Range	Standard Dev.
	Original EGD	5377	5865	5598.5	488	198.9630
Move and Swap	First Third	5347	5913	5608.2	566	185.3788
	Second Third	5424	5986	5619.1	562	186.6140
	Third Third	5491	5844	5632	353	135.4269
	Variable	5316	5750	5578.5	434	131.2210
	Full List	5384	5729	5600.3	345	120.4667
	Sequential	5362	5926	5642.2	564	179.8072
	Sequential Re-sort	5298	5767	5558.9	469	133.5693
	Sequential Reverse	5155	5724	5469.3	569	159.8194
	Sequential Reverse Re-sort	5311	5888	5555.9	577	186.6687

Table 2b - Full results for Exam 1 for the Move and Swap heuristic

Tables 2a and 2b provide an overview of results for Exam 1. Taking into account only the Move Only heuristic, the Sequential Re-sort method provides the lowest average penalty, with all other values beating that of the original EGD. When considering the Move and Swap heuristic, this result changes, such that the Sequential Reverse method now produces the best results. When comparing the results as a whole, Sequential Reverse on Move and Swap is overall the most competitive method. However while there are other selection techniques that are able to produce more consistent results as measured by the standard deviation, these may not necessarily be producing the best quality results as measured by soft constraint violations.

	Best Results				
	Worst	Best	Average	SD	Method
Exam 1	5155	5724	5469.3	159.8194	Sequential Reverse (MS)
Exam 2	430	465	449.7	11.324	Sequential Reverse Re-sort (MS)
Exam 3	10236	10909	10444.4	201.5403	Original EGD
Exam 4	19171	24405	20241	1516.1996	Original EGD
Exam 5	3077	3339	3184.6	79.6244	Sequential Reverse (MO)
Exam 6	25955	26320	26150.5	117.0102	Sequential Reverse Re-sort (MS)
Exam 7	4475	4688	4567.8	60.1956	Original EGD
Exam 8	7734	8317	8081.5	165.8294	Sequential Reverse (MS)
Exam 9	1101	1033	1060.6	23.7777	Sequential Resort (MS)
Exam 10	15809	14685	15293.9	333.3108	Sequential Reverse Re-sort (MO)
Exam 11	82721*	32175	44820.2	30709.12055	Sequential Resort (MO)
Exam 12	5694	5278	5464.5	132.7715	Sequential Reverse Re-sort (MO)

Table 3 - Best results

Table 3 identifies the best of the selection techniques studied and the neighborhood search heuristic where appropriate. Two of the new selection criteria, with the majority using the Move and Swap heuristic, outperform the original EGD algorithm in nine out of the twelve datasets. Both of the new selection criteria work on the principle of attempting to optimize the examination with the lowest weighted penalty first, working under the assumption that these exams would have a greater freedom of movement across the neighborhood. Exams 3, 7 and 11 are arguably the largest examinations within the set and contain the highest number of students. The two selection criteria that performed well on the other datasets may have spent a comparatively longer period iterating through examinations for which there was no available optimal move or swap during that iteration. This may also be the case with the other datasets, but to a lesser degree due to their smaller size. Exam 4 has a high conflict density at 15%, and a limited number of available time and room combinations. As such sequentially utilizing the whole weighted list may again waste computation time as comparatively fewer move or swap actions are available. While Exam 12 has a higher conflict density at 18.45% there is a greater number of available room and time combinations. The issues with Exams 3, 4, 7, and 11 may be solved by restricting what portions of the weighted list are used during optimization.

	Selected Method Best		Absolute Best		Other Techniques		
	Best	Average	Best	Average	Müller - ITC 2007 [20]	Graph Colouring [21]	Multistage Algorithmic [22]
Exam 1	5155	5469.3	5155	5469.3	**4370**	6234	5814
Exam 2	430	449.7	421	450.3	400	**395**	1062
Exam 3	10236	10444.4	**9827**	10304.3	10049	13002	14179
Exam 4	19171	20241	**17296**	19538.1	18141	17940	20207
Exam 5	3077	3184.6	**2971**	3166.7	2988	3900	3986
Exam 6	25955	26150.5	**25690**	26096	26585	27000	27755
Exam 7	4475	4567.8	4364	4505.7	**4213**	6214	6885
Exam 8	**7734**	8081.5	**7734**	8081.5	7742	8552	10449
Exam 9	1033	1060.6	**999**	1065.6	1030	N/A	N/A
Exam 10	14685	15293.9	**14562**	15293	16682	N/A	N/A
Exam 11	32175	44820.2	**30842**	68246.6	34129	N/A	N/A
Exam 12	5278	5464.5	**5256**	5477.7	5535	N/A	N/A

Table 4 - Comparison of best results and other techniques

Table 4 compares the best results of the best identified selection criteria, the best overall results generated with the modified code and those of Müller and recently published research utilizing the competition rules for time limits. Müller's four phased approach continues to show its strengths here, producing some of the lowest penalties of all the approaches listed. When comparing the best scores of the selected best selection techniques, the DEGD algorithm is able to outperform the other two techniques from literature, across six of the eight datasets and is also able to remain competitive with the approach taken by Müller. When comparing the best overall scores generated with any of the new selection techniques the DEGD algorithm is able to outperform Müller's technique in five of the eight datasets. Due to the prior unavailability of the hidden datasets, comparison results are not available for research that has been published post competition.

7 Conclusion and future work

This paper has presented a number of new selection techniques that can successfully utilize information gathered during Adaptive construction to direct the selection of the examination used in the neighborhood search process, in solving the Examination Timetabling problem as described in the 2nd International Timetabling Competition. The techniques presented here were not tailored for specifically solving this problem and the techniques within could potentially be adapted for other problem areas. In addition to testing against other benchmark datasets, the effectiveness of the technique to solve other timetabling and scheduling problems will be investigated in future works.

Utilizing the whole of the weighted list produced favorable results for the majority of the competition datasets, however the original technique was able to outperform on the largest and most complex datasets. Further work will look into identifying the best portion of the weighted list to use during optimization. By limiting the portion of the weighted list used sequentially to the areas that show the greatest freedom of movement, it is hoped that this situation will be rectified. However care will need to be taken to ensure that the same sections of the neighborhood are not repeatedly used during the search process. It is probable that the portion size of the weighted list that is optimal for use will be dependent on the structure of the exam data supplied. If this is the case, any future implementation would need to include self-adjusting parameters for the use of the weighted list, to ensure that the solver remains as general as possible. Another simpler approach to this would involve reinforcement learning, choosing the best performing portion of the weighted list.

Late Acceptance strategies are another interesting area of exploration. Work on this technique has already shown promising results, however there are issues when combined with Reinforcement Learning techniques as the delay within the acceptance mechanism causes the learning mechanism to respond less rigorously than expected to changes in the search environment.

It should be pointed out again that the results here are in compliance with the rules for the 2007 International Timetabling Competition. In future comparisons it is proposed that various time limits be used, to allow analysis of the effectiveness of the techniques over a longer time scale.

References

1. B. McCollum, "A perspective on bridging the gap between theory and practice in university timetabling," *Practice and Theory of Automated Timetabling VI*, vol. 3867, pp. 3–23, 2007.
2. B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. D. Gaspero, R. Qu and E. K. Burke, "Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition," *INFORMS Journal on Computing*, vol. 22, no. 1, pp. 120–130, May 2009.

3. M. W. Carter, G. Laporte and S. Y. Lee, “Examination Timetabling : Algorithmic Strategies and Applications,” *The Journal of the Operational Research Society*, vol. 47, no. 3, pp. 373–383, 1996.
4. R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot and S. Y. Lee, “A survey of search methodologies and automated system development for examination timetabling,” *Journal of Scheduling*, vol. 12, no. 1, pp. 55–89, Oct. 2008.
5. P. Mcmullan, “An extended implementation of the great deluge algorithm for course timetabling,” *Computational Science–ICCS 2007*, vol. 4487, pp. 538–545, 2007.
6. B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke and S. Abdullah, “An extended great deluge approach to the examination timetabling problem,” *Proceedings of the 4th Multidisciplinary International Conference on Scheduling: Theory and Applications*, no. August, pp. 424–434, 2009.
7. E. K. Burke, G. Kendall, B. McCollum and P. Mcmullan, “Constructive versus improvement heuristics: an investigation of examination timetabling,” *3rd Multidisciplinary International Scheduling Conference: Theory and Applications*, pp. 28–31, 2007.
8. E. Burke and J. Newall, “Solving Examination Timetabling Problems through Adaption of Heuristic Orderings,” *Annals of Operations Research*, vol. 129, no. 1–4, pp. 107–134, Jul. 2004.
9. T. B. Cooper and J. H. Kingston, “The Complexity of Timetable Construction Problems,” *Lecture Notes in Computer Science*, vol. 1153, pp. 281–295, 1996.
10. M. Dorigo, G. Di Caro and L. M. Gambardella, “Ant Algorithms for Discrete Optimization,” *Artificial Life*, vol. 5, no. 2, pp. 137–172, Apr. 1999.
11. M. Eley, “Ant algorithms for the exam timetabling problem,” *Practice and Theory of Automated Timetabling VI*, vol. 3867, pp. 167–180, 2006.
12. N. Pillay and W. Banzhaf, “An informed genetic algorithm for the examination timetabling problem,” *Applied Soft Computing*, vol. 10, no. 2, pp. 457–467, Mar. 2010.
13. E. Ozcan, Y. Bykov, M. Birben and E. K. Burke, “Examination timetabling using late acceptance hyper-heuristics,” in *2009 IEEE Congress on Evolutionary Computation*, 2009, pp. 997–1004.
14. E. Özcan, M. Misir, G. Ochoa and E. K. Burke, “A Reinforcement Learning - Great-Deluge Hyper-Heuristic for Examination Timetabling,” *International Journal of Applied Metaheuristic Computing*, vol. 1, no. 1, pp. 39–59, Jan. 2010.
15. R. Bai, J. Blazewicz, E. K. Burke, G. Kendall and B. McCollum, “A simulated annealing hyper-heuristic methodology for flexible decision support,” *4OR*, vol. 10, no. 1, pp. 43–66, Nov. 2011.
16. E. Ikonomovska, I. Chorbev, D. Gjorgjevik and D. Mihajlov, “The Adaptive Tabu Search and Its Application to the Quadratic Assignment Problem,” in *9th International Multiconference Information Society 2006*, 2006, pp. 26–29.
17. S. Abdullah and H. Turabieh, “On the use of multi neighbourhood structures within a Tabu-based memetic approach to university timetabling problems,” *Information Sciences*, vol. 191, pp. 146–168, May 2012.
18. G. Dueck and T. Scheuer, “Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing,” *Journal of Computational Physics*, vol. 90, no. 1, pp. 161–175, Sep. 1990.
19. D. P. Clements and D. E. Joslin, “Squeaky Wheel Optimization,” *Journal Of Artificial Intelligence Research*, vol. 10, pp. 353–373, May 1999.
20. T. Müller, “ITC2007 solver description: a hybrid approach,” *Annals of Operations Research*, vol. 172, no. 1, pp. 429–446, Oct. 2009.
21. N. R. Sabar, M. Ayob, R. Qu and G. Kendall, “A graph coloring constructive hyper-heuristic for examination timetabling problems,” *Applied Intelligence*, vol. 37, no. 1, pp. 1–11, Aug. 2011.
22. C. Gogos, P. Alefragis and E. Housos, “An improved multi-staged algorithmic process for the solution of the examination timetabling problem,” *Annals of Operations Research*, vol. 194, no. 1, pp. 203–221, Feb. 2010.

MISTA 2013

Evaluation of appointment scheduling rules

Creemers Stefan · Colen Pieter · Lambrecht Marc

1 Introduction

Professionals in healthcare and other services face the problem of allocating time windows to customers. This allocation can be done by means of so-called "appointment scheduling rules" (ASR). ASR determine when a customer receives service during a service session. Although the literature on ASR is mainly focussed on healthcare, the research topic is generic and applicable in many industries: attorneys, faculty receiving students, barbers, automobile service centers and many others.

Conducting scheduled appointments on time is becoming ever more important across service industries. Timeliness of appointments is a key concern both for patients seeking treatment [1] and for customers who wait for field service [2]. The customer waiting time consequently is a relevant performance measure. A second important objective of appointment scheduling has to do with the efficiency of the service. For private companies, the impetus to efficiency comes naturally. But also healthcare systems are under pressure to use their capacity effectively and efficiently. Doctors' (or more general servers') idle time and overtime are hereby important performance measures. The objective of this article is to identify ASR that minimize customer waiting time, server idle time and overtime. This has to be done in an environment where both demand and supply characteristics are highly uncertain and subject to many sources of variability.

ASR determine the planned (scheduled) arrival rate of customers during a service session. The actual arrival time may of course be different from the planned arrival time. We therefore assign each customer a probability of being too late or too early. In addition, we fix for each customer a probability of not showing up. The performance of

Creemers Stefan
IESEG School of Management, France
E-mail: s.creemers@ieseg.fr

Colen Pieter
Faculty of Business and Economics, Research Center for Operations Management, KU Leuven, Belgium
E-mail: pieter.colen@kuleuven.be

Lambrecht Marc
Faculty of Business and Economics, Research Center for Operations Management, KU Leuven, Belgium
E-mail: marc.lambrecht@kuleuven.be

ASR is not only influenced by the arrival rate and service rate characteristics. Other types of outages during the service session are also important. We therefore allow for delays at the start of a service session due to late arrival of the doctor or due to setup activities at the start of a session. We also allow preemptive and non-preemptive interrupts during the service session. All these extensions allow us to model real-life appointment systems and to identify ASR that have a robust performance across different settings.

We develop an analytical model that uses an efficient (in terms of computational and memory requirements) algorithm to assess the performance of ASR. The validity and accuracy of the model are supported by a simulation study. We use the model to assess the performance of a set of 314 ASR in an elaborate computational experiment. To compare the performance of these ASR (in terms of waiting time, idle time and overtime), we apply a data envelopment analysis (DEA).

2 Model Description

We focus on static ASR (i.e., decisions are made prior to the start of the service session). The Markov model used to evaluate the performance of an ASR has the following properties:

- Customers are served by a single server.
- Customers have i.i.d. service time distributions.
- All customers that arrive during the service session receive service.
- Customers that arrive early (i.e. prior to their scheduled arrival time) receive service if the server is idle. Note that this implies the possibility of overtaking other customers.
- The arrival process can differ for each individual customer

Contrary to other existing models, we allow for an individual characterization of the arrival process for each customer. In addition, computational performance and model accuracy (and hence practical applicability) of our model significantly exceed the capabilities of comparable models in the literature on appointment systems. The performance measures of interest are: (1) the expected customer waiting time; (2) the expected amount of server overtime; (3) the expected amount of server idle time. Our model may be used to obtain these performance measures for any given schedule of customers. However, we limit ourselves to the comparison of the performance of schedules that are generated by a set of 314 ASR.

In general, ASR are classified in terms of:

- A_i, the scheduled arrival time of customer i,
- μ^{-1}, the mean service time requirement of a customer,
- σ_i, the standard deviation of the service time requirement of customer i,
- N, the number of customers that require scheduling.

We implement a set of 314 ASR and use an analytical model to perform an extensive computational experiment in which the performance of these rules is assessed with respect to the three performance measures in a wide variety of settings. The adopted set of ASR is an extension of the set of 50 ASR selected in [3,4]. These ASR are common in daily practice or have been shown to yield good and robust results [3,4].

The ASR may be summarized as variations of: (1) the individual appointment rule; (2) the block appointment rule; (3) early-lateness rules (hereafter referred to as the EL rules).

The individual appointment rule schedules the arrival times of customers as follows:

$$\begin{aligned} A_i &= ia\mu^{-1} && \forall i : i < l, \\ A_i &= A_{i-1} + \mu^{-1} + h\sigma_i && \forall i : l \leq i < N. \end{aligned} \tag{1}$$

Where: (1) a is a multiplier to delay the start of the first arriving customers; (2) l denotes the number of customers scheduled for arrival at the start of a service session; (3) h is a multiplier used to adjust the impact of σ_i. We implemented 91 variants of the individual appointment rule by allowing parameters a, l and h to vary over set $\{0, 0.3, 0.5\}$, set $\mathbf{L} = \{1, 2, 3, 4, 5\}$ and set $\mathbf{H} = \{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$ respectively.

The block appointment rule may be summarized as follows:

$$\begin{aligned} A_i &= 0 && \forall i : i < b, \\ A_{nb} &= A_{(n-1)b} + b\mu^{-1} + h\sqrt{b\sigma_{nb}} \\ &\qquad \forall n : 1 \leq n < \tfrac{N}{b}, \\ A_{nb+i} &= A_{nb} \\ &\qquad \forall i : 1 \leq i < b. \end{aligned} \tag{2}$$

Where b denotes the block size (i.e. the number of customers assigned to arrive at a single time instance). Varying parameters b and h over set $(\mathbf{L} \setminus \{1\})$ and set $\mathbf{H}$ respectively, we obtain 28 ASR.

The EL rules speed up and/or slow down the pace of scheduled arrivals using correction factors r_1 and r_2. The computation of scheduled arrival times is performed in two steps. First, all scheduled arrival times are initialized using the individual appointment rule where $(l = 1)$ and $(h = 0)$. Next, a correction is applied to speed up and/or slow down the pace of scheduled customer arrivals.

$$\begin{aligned} &\text{Initialization:} \\ &A_0 = 0, \\ &A_i = A_{i-1} + \mu^{-1} \ \forall i : 1 \leq i < N. \\ &\text{Correction:} \\ &A_i = A_i - r_1(z - i)h\sigma_i \ \forall i : 1 \leq i \leq z, \\ &A_i = A_i - r_2(z - i)h\sigma_i \ \forall i : z < i < N. \end{aligned} \tag{3}$$

Where: (1) r_1 and r_2 are correction factors used to speed up or slow down the succession of scheduled arrivals; (2) z is any multiple of 5 smaller than N. Parameter r_1 controls the arrival pace of the first z customers; the arrival pace of these customers increases as r_1 increases. Conversely, parameter r_2 controls the arrival pace of those customers that are scheduled to arrive after customer z; the arrival pace of these customers decreases as r_2 increases. When varying parameter h over set $(\mathbf{H} \setminus \{0\})$ and parameters r_1 and r_2 over the set $\{0, 1, 2\}$ (where $(r_1 + r_2) > 0$), we obtain 39 times times $\left\lfloor \frac{N-1}{5} \right\rfloor$ ASR.

A summary of the ASR is given in table 2. We determine the performance of the ASR in a wide range of environmental settings. In the most simple environment, all customers arrive punctually at their assigned appointment time. Complexity is introduced in the form of so-called "environmental variables". An extensive overview of these environmental variables is provided in [5]. We take the following environmental variables into account:

Rule	$A_i = ia\mu^{-1}, \quad \forall i : i < l,$
	$A_i = A_{i-1} + \mu^{-1} + h\sigma_i, \quad \forall i : l,$
Rule no.	$1-7, 8-14, 15-21, 22-28, 29-35,$
Conditions	$l = 1, 2, 3, 4, 5 \wedge a = 0 \wedge h \in \mathbf{H},$
Rule no.	$36-42, 43-49, 50-56, 57-63,$
Conditions	$l = 2, 3, 4, 5 \wedge a = 0.3 \wedge h \in \mathbf{H},$
Rule no.	$64-70, 71-77, 78-84, 85-91,$
Conditions	$l = 2, 3, 4, 5 \wedge a = 0.5 \wedge h \in \mathbf{H},$
Rule	$A_i = 0, \quad \forall i : i < b,$
	$A_{nb} = A_{(n-1)b} + b\mu^{-1} + h\sqrt{b\sigma_{nb}}, \quad \forall n : 1 \leq n < \frac{N}{b},$
	$A_{nb+i} = A_{nb}, \quad \forall i : 1 \leq i < b,$
Rule no.	$92-98, 99-105, 106-112, 113-119,$
Conditions	$b = 2, 3, 4, 5 \wedge h \in \mathbf{H},$
Rule	initialize$A_0 = 0, \quad A_i = A_{i-1} + \mu^{-1}, \quad \forall i : 1 \leq i < N,$ then
	$A_i = A_i - r_1(z-i)h\sigma_i, \quad \forall i : 1 \leq i \leq z,$
	$A_i = A_i - r_2(z-i)h\sigma_i, \quad \forall i : z < i < N,$
Rule no.	$120-125, 159-164, 198-203, 237-242, 276-281,$
Conditions	$z = 5, 10, 15, 20, 25 \wedge r_1 = 0 \wedge r_2 = 1 \wedge h \in (\mathbf{H} \setminus \{0\}),$
Rule no.	$126-128, 165-167, 204-206, 243-245, 282-284,$
Conditions	$z = 5, 10, 15, 20, 25 \wedge r_1 = 0 \wedge r_2 = 2 \wedge h \in \{0.2, 0.25, 0.3\},$
Rule no.	$129-134, 168-173, 207-212, 246-251, 285-290,$
Conditions	$z = 5, 10, 15, 20, 25 \wedge r_1 = 1 \wedge r_2 = 0 \wedge h \in (\mathbf{H} \setminus \{0\}),$
Rule no.	$135-140, 174-179, 213-218, 252-257, 291-296,$
Conditions	$z = 5, 10, 15, 20, 25 \wedge r_1 = 1 \wedge r_2 = 1 \wedge h \in (\mathbf{H} \setminus \{0\}),$
Rule no.	$141-146, 180-185, 219-224, 258-263, 297-302,$
Conditions	$z = 5, 10, 15, 20, 25 \wedge r_1 = 1 \wedge r_2 = 2 \wedge h \in (\mathbf{H} \setminus \{0\}),$
Rule no.	$147-149, 186-188, 225-227, 264-266, 303-305,$
Conditions	$z = 5, 10, 15, 20, 25 \wedge r_1 = 2 \wedge r_2 = 0 \wedge h \in \{0.2, 0.25, 0.3\},$
Rule no.	$150-155, 189-194, 228-233, 267-272, 306-311,$
Conditions	$z = 5, 10, 15, 20, 25 \wedge r_1 = 2 \wedge r_2 = 1 \wedge h \in (\mathbf{H} \setminus \{0\}),$
Rule no.	$156-158, 195-197, 234-236, 273-275, 312-314,$
Conditions	$z = 5, 10, 15, 20, 25 \wedge r_1 = 2 \wedge r_2 = 2 \wedge h \in \{0.2, 0.25, 0.3\}.$

Table 1 Summary of the different appointment scheduling rules

- The number of customers served during a service session.
- The customer service time.
- Customer unpunctuality (i.e., the early/late arrival of customers).
- Customer no-shows.
- Late start of the service session.
- The variability of service times and customer unpunctuality.

By combining different values for the environmental parameters, we obtain 243 environmental settings in which we evaluate the performance of the ASR.

3 Results

After obtaining the relevant performance measures by means of the Markov model, we perform a DEA to get a performance ranking of all the ASR. This ranking enables us to select those ASR that work best under a wide range of settings and performance criteria. In order to assess the performance of an ASR, we use a composite indicator (CI):

$$\mathrm{CI}_i = \frac{1}{\sum_{k=1}^{3} w_{i,k} x_{i,k}}, \tag{4}$$

Rank	ASR	CI (%)	Type	Weight sensitivity
1	15	99.9980	Individual	0.195624
2	22	99.9976	Individual	0.313921
3	30	99.9964	Individual	0.389075
4	23	99.9954	Individual	0.284273
5	29	99.9941	Individual	0.412453
6	16	99.9838	Individual	0.163625
7	273	99.9724	EL	0.066703
8	162	99.9650	EL	0.171877
9	256	99.9453	EL	0.070306
10	298	99.9184	EL	0.064002

Table 2 Ranking of ASR based on average efficiency across environments

where: (1) $w_{i,k}$ is the weight assigned to performance measure k for ASR i and (2) $x_{i,k}$ is the score of ASR i for performance measure k. Weights $w_{i,k}$ are determined by means of a DEA. We use the Maverick index to assess the sensitivity of a CI-score with respect to the chosen weight set (refer to [6] for further details).

Table 2 lists the best performing ASR. We report the average CI values across the different environmental settings. In the table we also indicate the type of ASR (i.e., individual, block or EL) and the sensitivity of the chosen weight set (i.e., the Maverick index). It is striking that the six best performing ASR are all individual ASR with the common characteristics of zero delay for the first arrival, an initial number of patients of three or more and with a very small to zero adjustment for service time variance. This observation is encouraging for practitioners as these rules are simple to implement. At the downside, the maverick index indicates that the efficiency scores of these simple ASR are quite sensitive to the weight selection. The lower sensitivity to the weight selection can be a reason to opt for an EL rule, which are firmly established in the top 15 as from rank seven. Block ASR are clearly the less attractive type of ASR. The best performing block ASR have a block size of two with a small or zero adaption for service time variance. These findings corresponds with the conclusions of earlier research [7,3]. Lastly, we note that the simple Bailey-Welch rule with two initial arrivals followed by one-by-one arrivals (ASR 8) performs rather well both in terms of the efficiency score and weight sensitivity (respectively 99,81% and 0,0959).

We can conclude that individual ASR have the best performance across environments taking into account server overtime, idle time as well as customer waiting time. EL ASR, however, seem to be more robust with respect to the value judgement (i.e., the importance assigned to each of the performance measures). The performance of block ASR is dismal and they should be avoided because better alternatives clearly exist.

References

1. Grote, K.D., Newman, J.R. and Sutaria, S.S., A better hospital experience, The McKinsey Quarterly, November, 1-11 (2007)
2. Apte, A, Apte, U. M. and Venugopal, N., Focusing on customer time in field service: A normative approach, Production and Operations Management, 16(2), 189-202 (2007)
3. Ho, C. J. and Lau, H. S., Minimizing total-cost in scheduling outpatient appointments, Management Science, 38(12), 1750-1764 (1992)
4. Ho, C. J. and Lau, H. S., Evaluating the impact of operating conditions on the performance of appointment scheduling rules in service systems, European Journal of Operational Research, 112(3), 542-553 (1999)

5. Cayirli, T. and Veral, E., Outpatient scheduling in health care: a review of literature, Production and Operations Management, 12(4), 519-549 (2003)
6. Doyle, J. and Green, R., Efficiency and cross-efficiency in DEA - derivations, meanings and uses, Journal of the Operational Research Society, 45(5), 567-578 (1994)
7. Welch, J.D., Appointment systems in hospital outpatient departments, Operations Research Quarterly, 15(3), 224-232 (1964)

MISTA 2013

In-depth analysis of the quality sequence to resolve CRSP

Valero-Herrero, Maria • Garcia-Sabater, Jose P. • Vidal-Carreras, Pilar I.

Abstract The automotive industry has employed mixed-model assembly lines to obtain efficient production flows despite working to a very diverse portfolio of products. In this context, the sequence in which products appear on the line is of utmost importance. Although the sequence is contemplated at the beginning of the production process, perturbations in production frequently occur in the sequence throughout the process. A situation may also arise in which a sequence required by a process differs from that required by a former process. Hence, resequencing is necessary and can be carried out by using buffers. This work addresses the resequencing version of car sequencing (CRSP) with selectivity banks. Buffer input and output algorithms are proposed, which are verified by the simulation models implemented in SIMIO. A sensitivity analysis is also done for the different operational factors.

1 Introduction

Mixed-model assembly lines allow the efficient production of a large quantity of different models. For this reason, they are employed in many industries, especially in car manufacturing. The use of these lines requires overcoming the short-term sequencing problem, to which the literature has paid considerable attention [1]. The objective of the sequencing problem is to determine the succession order of models appearing on the line by considering different objectives such as components consumption levelling, levelling the appearance of models on the line, workloads on stations and total line stoppage time [2]. The components levelling is important for both internal [3] and external logistics. In [4;5] is showed the effects of suppliers sequence.

As suggested by Boysen et al. in [6], the problem considered in practical applications tends to be a resequencing problem in which reordering the appearance sequence is sought rather than the sequencing problem as set out in most works in the literature.

On automobile assembly lines, resequencing is necessary for two main reasons. Firstly, the different sections making up the process (the body shop, the paint shop, the assembly shop) have different sequencing objectives [7]. Secondly, a series of unforeseen events may occur, such as breakdowns, defects, shortages of material, which amend the initial sequence considered. Other causes can be given to the rupture of the sequence as the regulation buffers

existence or existence of parallel lines where products are branched and reintroduced on a line later [8].

For resequencing, different types of buffers may be employed:

- An Automated Storage and Retrieval System (AS/RS) is a buffer with numerous locations where each location is accessible. Thus this type of buffers offers more flexibility when it comes to resequencing. In the automobile industry, AS/RS tend to be used as Painted Body Storage (PBS), as in [9;10].
- Buffers consisting in multiple pull-off tables are placed alongside the line. A product can be extracted from a line by storing it on the pull-off table so that the following models advance to this position and the product is re-inserted into a later position in the sequence. These buffer applications are located opposite the paints section to construct subsequences with the same colour [11;12].
- Parallel line buffers consist in multiple lines where products are placed in a queue and become accessible for the buffer to extract the first product on each line.

This work deals with the resequencing problem, the Car Resequencing Problem – CRSP [13], which is considered in a parallel line buffer and is situated between the interior covering assembly lines (trim) and the final assembly line into which the motor part of vehicles is incorporated in an automobile manufacturing plant.

According to the classification established by Boysen et al. [6] for resequencing problems (Figure 1), the problem contemplated in the present work can be considered physical operational (op) and dynamic (dy) resequencing, because decisions are made in real time with the information available at all times. It can also be classified as reactive and proactive (re+pro) resequencing; proactive because it is performed in a buffer situated between two sections requiring different sequences; reactive because it may be due to perturbations in the sequence caused by breakdowns in former sections, shortages of material, urgent orders and defects. Regarding configuration, it is a parallel line buffer (mix), and the resequencing objective would be Car Sequencing (car), and avoiding disregarding the constraints fixed by the assembly line by employing sequencing rules.

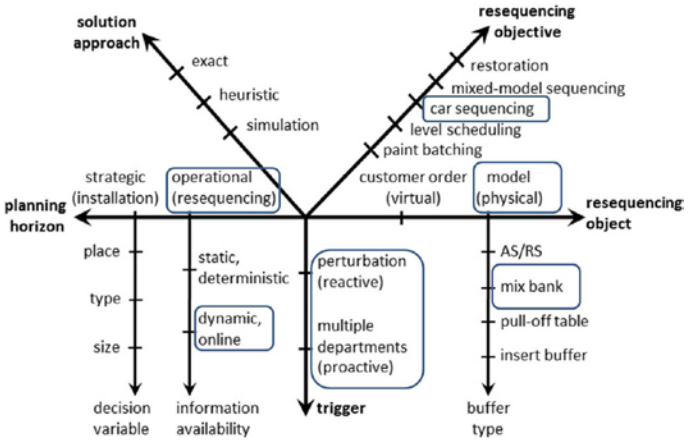


Fig. 1. Graph depicting the resequencing problems classification [6].

This work is arranged as follows: having presented the literature on the use of parallel lines buffers, there is a description of how the buffer operates. Then Section 4 describes the contemplated model for resequencing by using the algorithms implemented in buffer inputs and outputs. Section 5 presents the simulation model implemented in SIMIO. Finally, Section 6 provides the results of the experimentation done for the model performance study by amending the number of lines or by simulating stoppages in former buffer sections and the conclusions drawn from this work.

2 Resequencing by a parallel line buffer

Parallel line buffers, also called mix banks or selectivity banks in the literature, consist in several parallel lines with a limited capacity (normally the same for all lines) which are used to store products. Each line is arranged as if in a queue, products are introduced into the lines with available capacity and products are extracted from those situated at the front (the first position) of each line.

The literature contains different applications of parallel line buffers which are situated before the paints section where the objective is to minimize changes in colour [14-22]. Nevertheless, there are very few works on resequencing by parallel lines before the assembly line. The objective of the majority of works involves reordering [14;23-26]; that is, reordering the sequence to obtain the initial sequence. Very few authors [27-30] have set the objective in sequencing of disregarding the constraints established by the assembly line as little as possible.

Resequencing in such buffers with several lines in which to store products has, in turn, two decision problems: buffer input and buffer output. In this sense, Moon [27] establishes the following buffer input and output criteria. First select the cars which exceed the maximum allowed time at the front of the line, and the maximum storage time allowed. Of these, see if any car disregards the constraints. Of those that do not, select the one with the lowest smooth ratio. The smoothing of components or models can be measured. Moon [27] only measures the smoothing of models and looks for a sequence in which each model is manufactured uniformly. If there is no car that does not disregard any constraint, make a selection according to the time it is stored in the buffer.

Choi and Shin [28] employ lines that are assigned to certain products. If the product has no assigned line or if the corresponding line is full, the product is introduced into the line whose last product is the least similar. For outputs, the intention is to minimize the number of constraints. If no constraint is disregarded, the product with less regularity within the buffer will be extracted.

3 A detailed description of the problem

The buffer that is the object of this study is a parallel line buffer (Fig. 2), which is situated between the trim assembly lines and the final assembly line. This buffer is employed to compensate the stoppages that arise in various assembly line areas. As it has parallel L lines, we can resequence because there are as many vehicles available as the number of buffer lines at all times. Extraction candidates will be located at the front of each line.



Fig. 2. A parallel L line buffer (see Table 1).

Each vehicle will be assembled according to customer specifications, who will have previously defined them depending on whether or not a specific option (k) is necessary. The Vijk property of a product located at position j on line i takes a value of 1 if it requires option k, and a value of 0 if not. If R options, which influence the assembly time, are involved, the total number of possible models will be R^2.

For each option, which assumes a spacing constraint, also known as physical constraints [31], an M_k:N_k-type sequencing rule is available. The usual way to express these constraints, this being a generalisation of the proposal put forward by Monden [32], is: "No more k-type M cars for each consecutive N units". This same constraint can be applied as a control of the minimum which, despite not being associated with a physical constraint, can be used to guarantee regularity in the appearance of products and components. "No less k-type M cars for each consecutive N units". For all the rules, a Z_k penalisation applies in accordance with each rule's restrictiveness.

Information about both sequencing rules and the weight assigned to them is obtained from knowledge about the line and its performance. However, amendments can be made later, particularly if the mix of the production on the line is amended significantly.

4 Model for resequencing CSP in a parallel line buffer

4.1 Notation

The notation employed in developing the algorithms is presented in Table 1.

Table 1. Notation

Index	
i (L)	The index referring to buffer lines.
j (C)	The index referring to the position within each line, with the first position being closest to the output.
k (NR)	The index referring to product properties.
Parameters	
$V_{0,j,k}$	Value of property *k* of the product situated at position *j* of the line before the buffer. It takes value 1 if it has this property, and 0 otherwise.
$V_{i,j,k}$	Value of property *k* at position *j* of buffer line *i*. It takes value 1 if it has this property, and 0 otherwise.
$B_{j,k}$	Value of property *k* at position *j* of the buy-off, a line before the final assembly line. It takes value 1 if it has this property, and 0 otherwise.
L	Number of buffer parallel lines.
C	Capacity of each buffer line.
R	Number of constraints.
P	Depth. Number of units used to make comparisons.
Z_k	Weight associated with disregarding constraint *k*.
M_k:N_k	Sequencing rule. No more M_k products of the consecutive N_k with option *k*.
MIX_k	Proportion of products whose *k* property is equal to 1. It is calculated in relation to the total products. Data defined in the products input.
Variables	
q_i	Quantity of products on one line.
w_i	A value that quantifies the weight of each line to assign a product to a line.
s_{jk}	Value of property *k* at position *j* of the sequence to be evaluated.
n_i	Quantity of extracted product of one line during a simulation.
z	Total penalisation of a sequence.
z_p	Penalisation of a sequence with depth p.
z_{min}	Penalisation of the sequence with depth 1, whose sequence with depth p has penalisation minimum.
z_{minp}	Minimum penalisation of one sequence with depth p.
s	Sequence to be evaluated, with number line.
v	Sequence to be evaluated, with buffer position.
l	Line assigned

4.2 Input algorithm

The input algorithm is executed when a product is situated at position V_{01} (Figure 2) after deciding the line on which the product will be stored.

To place products in the buffer, the space available on each line and a criterion relating the line content with the product to be inputted are taken into account.

A weight per line is obtained in accordance with the decision rule considered. Finally, the line with a heavier weight will be assigned to the product:

$$l=i;\ w_i=\max(w_1,\ldots,w_L) \tag{1}$$

Next, details of each rule are provided independently, although combinations of them are possible.

Rule 1: The emptier line

The weight assigned to each line is inversely proportional to the quantity of products already on this line.

$$w_i = \frac{1}{q_i + 1} \tag{2}$$

Rule 2: The line more similar

The product about to be inputted is compared with the last one on the line for each property.

$$w_i = \frac{\sum_{k=1}^{R} V_{01k} \cdot V_{iq_ik}}{R} \tag{3}$$

This rule can apply to depth; that is to say, by comparing not only with the last product on the line. For example (Figure 3), if depth is considered to be 3, it is compared with the three last products on each line. If on any line the number of products is less than the depth value taken, it is compared with all the products on the line.

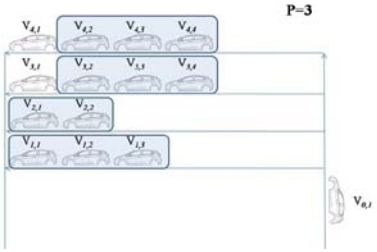


Fig. 3. Comparing the input product with all the lines by taking P=3.

Thus if compared with depth, the w_i value will be obtained as follows:

If $q_i \geq P$, then:

$$w_i = \frac{\sum_{k=1}^{R}\left(V_{01k} \cdot \sum_{q_i-P}^{q_i} \frac{V_{ijk}}{P}\right)}{R \cdot P} \tag{4}$$

If $q_i < P$ then:

$$w_i = \frac{\sum_{k=1}^{R}\left(V_{01k} \cdot \sum_{1}^{q_i} \frac{V_{ijk}}{P}\right)}{R \cdot q_i} \tag{5}$$

Rule 3: The more similar line by weighting according to the likelihood of disregarding and/or penalisation

Weighting can be achieved by bearing in mind the following variables and a combination of these variables:

- Penalisation assigned to disregarding a constraint. The higher the penalisation for disregarding a property, the heavier the weight assigned to the line whose last product has the same property.
- A product mix with this property. The larger the quantity of products a property has, the heavier the weight assigned to the line whose last product has this same property. The mix of the products inside the warehouse is considered, as Choi does in (1997).
- A Space among Products according to Constraints. If the constraints are of the type "No more M products of consecutive N units", then the heavier the weight the higher the M/N quotient.

As in the previous rule, it can be compared with depth.

4.3 Output algorithm

The output algorithm is executed every takt time, and it returns the line from which the following product will be extracted. To extract buffer products, the quality of the obtained sequence is taken into account. Sequence quality is measured according to the total penalisation obtained from disregarding constraints. The lower the penalisation obtained (z), the better this sequence is considered.

The quality sequence is in-depth analysed, that is, the available sequences after remove a product are also analysed. When depth is equal to 1, the sequences obtained from the extraction of one product are only analysed. In the case of a buffer with 4 lines (figure 4), the z value for each line is obtained by evaluating the obtained sequence (S) to add to the sequence the first product of that line.

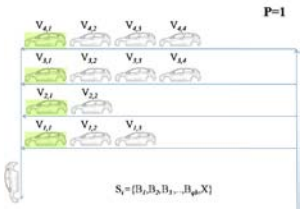


Fig.4. Available elements to be introduced into the sequence when depth P=1.

When P>1 is considered, the penalisation of a sequence would be also consider the penalisations of all previous sequences. For example, the penalisation of the sequence obtained from extract the second product from the line 1 (z_{21}), would be sum to the penalisation of the sequence obtained from extracting only the first product of the line (z_{11}). Also, the sequence with minimum penalisation is chosen. In figure 5 the available products after the extraction of the first product of each line are shown.



Fig. 5. Available elements to be introduced into the sequence when depth P=2.

The sequence obtained is evaluated applying for each constraint the equation (6):

$$\left(\sum_{j=1}^{N_k-1} S_{jk} + 1 > M_k\right) \wedge \left(V_{i1k} = 1\right) \Rightarrow z_i = z_i + Z_k \quad (6)$$

Now the logic of the output algorithm is described.

1. For each line (x), we see that there are enough pieces in the line, considering the virtually extracted pieces ($q_x > n_x$.) and add a piece to the sequence to evaluate S (v_{px}).

2. Sequence (S) is evaluated and the penalty of that sequence is obtained (z).

3. If the penalty is lower than the minimum penalty obtained with a sequence with p units ($z < z_{minp}$), we continue adding pieces to the sequence until to obtain a sequence of p pieces.

4. Finally, we take the sequence of p pieces with minimum penalisation and extract the first piece of that sequence.

Output Algorithm

1 Initialization: Set $z_{min}=M$; $z_{minp}=M$; $S=\{B_1,B_2,\ldots, B_{qb}\}$

2 **for** x=1 to L **do** [$n_x=0$; calcule q_x; actualise V_{xyk};]

3 **for** x=1 to L **do**

4 **if** ($q_x > n_x$) {Execute Evaluate(S) with $S=S+v_{px}$;$v_{px}=V_{xy}$; $y=n_x+1$;

5 $z_p = z$ (z=result of Evaluate(S)); $z = z_1 + \ldots + z_p$

6 **if** ($z < z_{minp}$) { $n_x = n_x+1$; p++;

7 **if** (p<P) {return to step 3}

8 **else {**l=x; $z_{min} = z_1$}

9 **}}return** l; z_{min};

5 Experimentation

Simulation was done in the SIMIO tool. Figure 4 provides a sample snapshot of the simulation model developed. The buffer configuration considered is based on a buffer installed in a Spanish plant of an automotive manufacturer. This buffer consists of 4 lines whose capacity is 20 vehicles to each line.

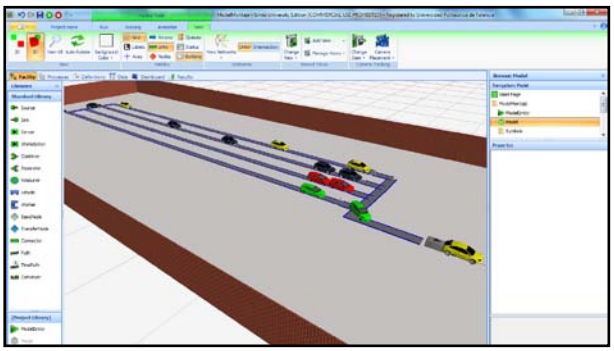


Fig. 4. Snapshot of the simulation model. http://www.youtube.com/watch?v=PxnZaAuQ0BE

5.1 Variation in the input algorithm

The input algorithms indicated in Table 2 were incorporated into the model according to the rules set out in the previous section.

Table 2. Input algorithms

Algorithm	Rule applied
A1	Emptier line
A2	More similar line P=1
A3	More similar line P=2
A4	More similar line P=3
A5	More similar line P=1 weighting by Z
A6	More similar line P=1 weighting by MIX
A7	More similar line P=1 weighting by M/N
A8	More similar line P=2 weighting by Z
A9	More similar line P=2 weighting by MIX
A10	More similar line P=2 weighting by M/N

5.2 Variation in the output algorithm

The output algorithms indicated in Table 3 were incorporated into the model according to the depth P established.

Table 3. Output algorithms

Algorithm	Depth applied
B1	PS=1
B2	PS=2
B3	PS=3

5.2 Type of models to produce and the constraints assigned

Tables 4 and 5 provides the constraints used in this study. The products of the assembly line considered in this article have six options that influence the assembly time and there are two scenarios modifying the mix of the options that have related with the stoppages and holdups.

Table 4. Spacing Constraints Scenario 1

Constraint k	M	N	W	MIX
R1	1	2	1000	0.3
R2	2	3	1000	0.2
R3	2	6	900	0.4
R4	1	7	400	0.2
R5	1	2	90	0.4
R6	1	2	1000	0.2

Table 5. Spacing Constraints Scenario 2

Constraint k	M	N	W	MIX
R1	1	2	1000	0.5
R2	2	3	1000	0.3
R3	2	6	900	0.4
R4	1	7	400	0.2
R5	1	2	90	0.4
R6	1	2	1000	0.2

5.3 Stoppages caused

In the experimentation work done, the following stoppages were simulated:

- Stoppages in previous lines. Stoppages in the trim A line were simulated. Trim A is associated to having the component 1.
- Holdups due to shortages of some material. Shortages of material on previous lines causes holdups in the models containing this component.

The simulated stoppages and holdups are shown in Table 6.

Table 6. Simulated stoppages and holdups

Cause	Duration (min)	Frequency (min)
Stoppage of trim A	5	100
	10	100
	15	100
Holdup as component 2 is lacking	10	200
	20	200
	30	200

5.4 Output data

Buffer performance is evaluated according to the sequence obtained on the final assembly line. In this work, the measure which is considered to measure sequence quality and, therefore, buffer performance is the total penalisation ratio. This indicator shows the mean of the total penalisation obtained through disregarding constraints.

6 Performance analysis

In the next figures the means of the ratio penalisations are showed for some of the different stoppages and holdups simulated. In order to show the performance of the input algorithm, in figure 5 the simulation results for the cases with output algorithm B1 and Scenario 1 are illustrated.

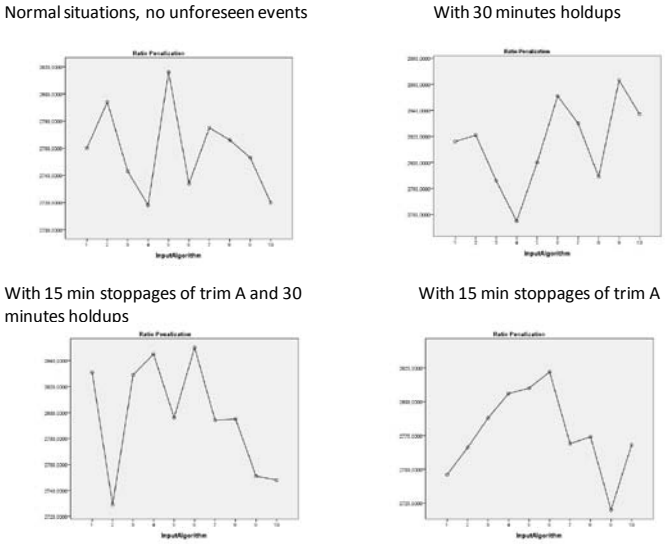


Fig. 5. Ratio Penalization for Scenario 1, Output Algorithm B1.

The simulation results obtained show that, depending on the given situation, the choice of the best input algorithm varies. Hence we observe that A4 (More similar line P=3) and A10 (More similar line P=2 weighting by M/N) perform better in normal situations when no unforeseen events (stoppages or shortages of material) arise. Similarly, algorithm A4 is the most appropriate for those situations in which a large quantity of products accumulates the same property subject to a constraint but is the less appropriate if there are also line stoppages.

For each situation, the input algorithm, the depth on the analysis of the output algorithm and the mix of the properties have been analised. In this analysis only the results for the A1 and A3 input algorithms are showed in order to illustrate the performance variation between two of them.

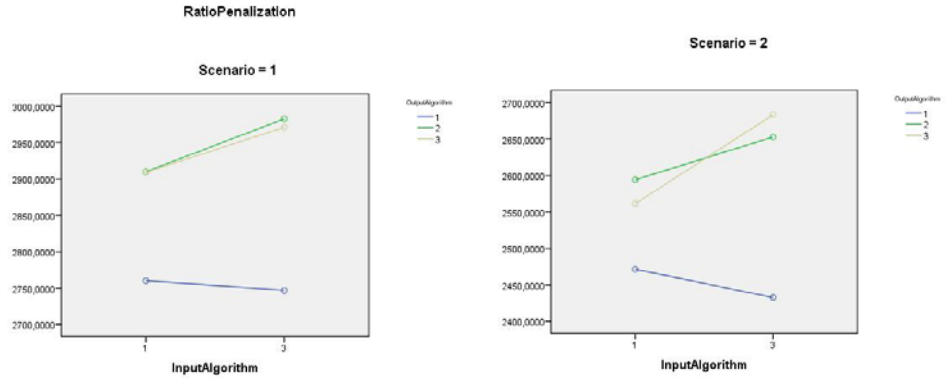


Fig. 6. Normal situations, no unforeseen events.

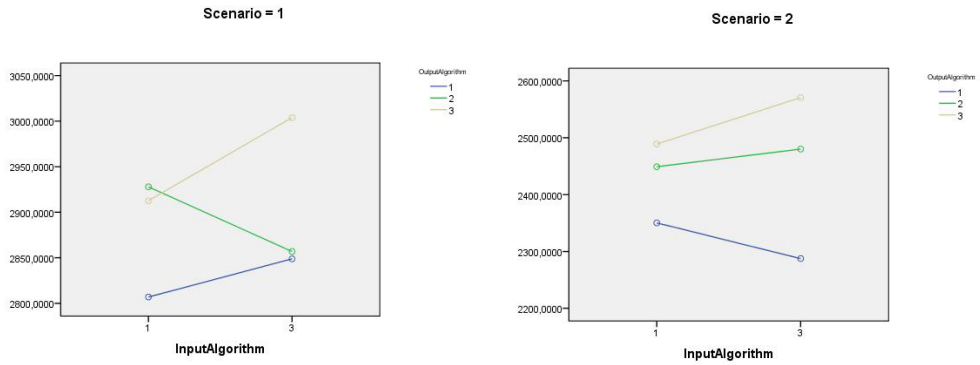


Fig. 7. With 30 minutes holdups.

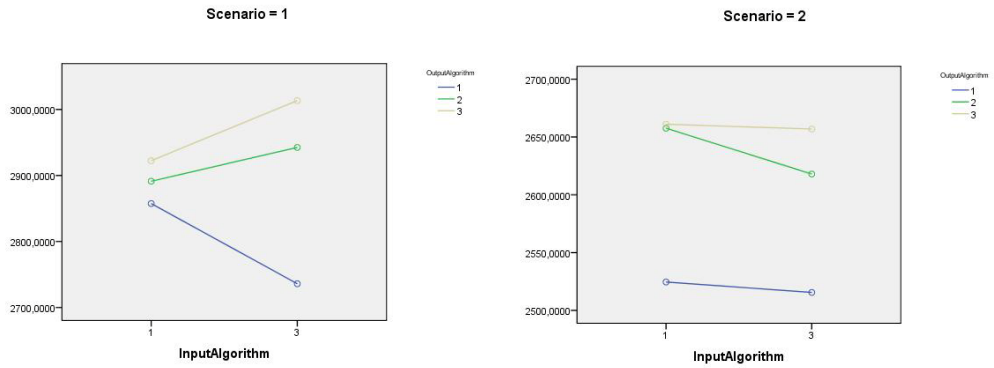


Fig. 8. With 15 min stoppages of trim A.

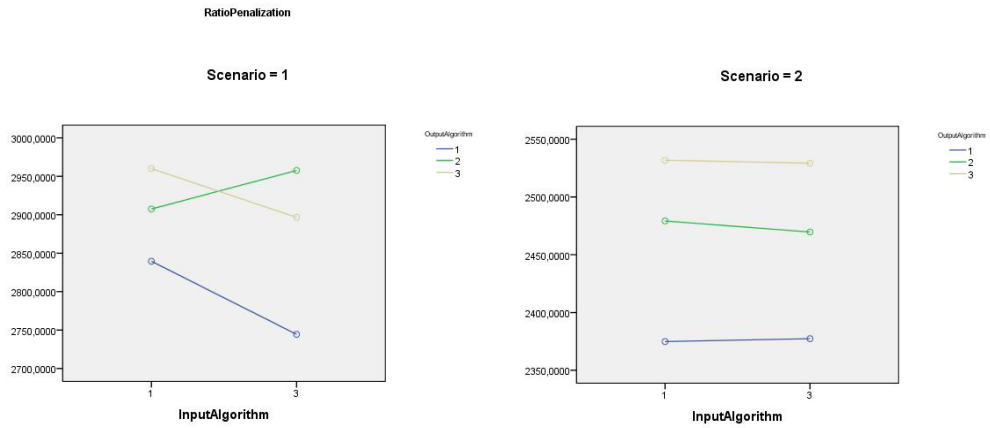


Fig. 9. With 15 min stoppages of trim A and 30 minutes holdups.

As can be seen in previous figures, the performance is different if the properties mix is different. With the analysed cases, can be concluded that the scenario 2, with a higher proportion of products with property 1 and 2, allow obtaining sequences with lower penalisations.

As for the in-depth analysis at the output buffer, the best results are obtained by analyzing with depth = 1. Sometimes, the analysis with depth 3 is better than the analysis with depth=2.

Nearly, the input algorithm that gets better results is the algorithm A3 (More similar line P=2)

7 Conclusions

The automotive industry has utilised mixed-model assembly lines where a wide variety of models is produced on the same assembly line. On this type of lines, sequencing systems dynamics substantially affects productivity on the line. Despite the importance of resequencing dynamics, the literature in this domain is scarce. This work presents a resequencing model for a parallel line buffer where it is developed and implemented.

The simulation results obtained show that, depending on the given situation, the choice of the best input algorithm varies.

Another important point raised is model performance when amending the operational conditions on the line. Different scenarios have been simulated with stoppages in former lines, a situation which actually occurs quite often. Different results were obtained for each algorithm in accordance with the variety of the models in the buffer, and this variation came about due to the stoppages occurring upstream of the buffer.

Hence we observe that A3 (More similar line P=2) and B2 (depth 2 in output analysis) perform better in normal situations when no unforeseen events (stoppages or shortages of material) arise. Similarly, algorithm A1 (Emptier line) and B2 (depth 2 in output analysis) is the most appropriate in some cases (scenario 1), for those situations in which a large quantity of products accumulates the same property subject to a constraint.

Future works are proposed to evaluate more scenarios and other configuration buffers for example modifying the number of lines. Furthermore, the in-depth analysis with higher depths would obtain different conclusions. Moreover, the development of a dynamic model is proposed that amends the algorithm applied according to the situation in the buffer, caused by stoppages or other unforeseen events, which vary the expected proportion of models.

Acknowledgements

The work described in this paper has been supported by the project "CORSARI MAGIC DPI2010-18243" by the Ministerio de Ciencia e Innovación del Gobierno de España within the Program de "Proyectos de Investigación Fundamental No Orientada".

This paper also has been written with financial support from the Project “Path Dependence y toma de decisiones para selección de herramientas y prácticas de Lean Manufacturing” (PAID-06-12-SP20120717) of the Universitat Politècnica de València.

References

1 Boysen N, Fliedner M, Scholl A: Sequencing mixed-model assembly lines: Survey, classification and model critique. European Journal of Operational Research 2009;192:349-373.
2 Xiaobo Z, Zhou Z, Asres A: A note on Toyota's goal of sequencing mixed models on an assembly line. Computers & Industrial Engineering 1999;36:57-65.
3 Valero-Herrero M, Garcia-Sabater JP, Morell-Monzo V: La alimentación a una línea de montaje: Procedimientos y Costes; 16th International Conference on Industrial Engineering and Industrial Management. XVI Congreso de Ingeniería de Organización. 2012.
4 Garcia-Sabater JP, Garcia-Sabater JJ, Marin-Garcia JA: Optimización de los recursos humanos a través de las TICs. Resultados de la reorganización laboral de un proveedor logístico. Economia y industrial 2008;370:143-151.
5 Garcia-Sabater JJ, Garcia-Sabater JP, Vidal-Carreras PI, Maheut J: Simulación de la entrega secuenciada de asientos en un proveedor de primer nivel del sector del automóvil; 2010.
6 Boysen N, Scholl A, Wopperer N: Resequencing of mixed-model assembly lines: Survey and research agenda. European Journal of Operational Research 2012;216:594-604.
7 Poler R, Garcia-Sabater JP, Rodriguez A, Lario FC: Un algoritmo greedy para la agrupación de colores en la secuencia maestra de una empresa de automoción; III Jornadas de Ingeniería de Organización. 1999.
8 García J, Rodriguez A, Poler R, Lario FC: Cumplimiento de la secuencia prevista y uso de almacenes reguladores; III Jornadas de Ingeniería de Organización. 1999, vol Barcelona.
9 Inman RR: ASRS sizing for recreating automotive assembly sequences. International Journal of Production Research 2003;41:847-863.
10 Gusikhin O, Caprihan R, Stecke KE: Least in-sequence probability heuristic for mixed-volume production lines. International Journal of Production Research 2008;46:647-673.
11 Lahmar M, Ergan H, Benjaafar S: Resequencing and feature assignment on an automated assembly line. Robotics and Automation, IEEE Transactions on 2003;19:89-102.
12 Gujjula R, Günther HO: Resequencing mixed-model assembly lines under just-in-sequence constraints; 2009, pp 668-673.
13 Boysen N, Golle U, Rothlauf F: The Car Resequencing Problem; Department of Information Systems and Business Administration, Johannes Gutenberg-University Mainz, 2010.
14 Ding FY, Sun H: Sequence alteration and restoration related to sequenced parts delivery on an automobile mixed-model assembly line with multiple departments. International Journal of Production Research 2004;42:1525-1543.
15 Han YH, Zhou C: Dynamic sequencing of jobs on conveyor systems for minimizing changeovers. The International Journal of Advanced Manufacturing Technology 2010;49:1251-1259.
16 Kittithreerapronchai O, Anderson C: Do ants paint trucks better than chickens? Markets versus response thresholds for distributed dynamic scheduling; Evolutionary Computation, 2003. CEC '03. The 2003 Congress on. 2003, pp 1431-1439.
17 Nouyan S: Agent-Based Approach to Dynamic Task Allocation Ant Algorithms; in Dorigo M, Di Caro G, Sampels M, (eds):Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2002, pp 43-99.
18 Spieckermann S, Gutenschwager K, Voss S: A sequential ordering problem in automotive paint shops. International Journal of Production Research 2004;42:1865-1878.
19 Campos M, Bonabeau E, Theraulaz G, Deneubourg JL: Dynamic scheduling and division of labor in social insects. Adaptive Behavior 2000;8:83-95.

20 Cicirello V, Smith S: Distributed Coordination of Resources via Wasp-Like Agents Innovative Concepts for Agent-Based Systems; in Truszkowski W, Hinchey M, Rouff C, (eds):Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2003, pp 71-80.
21 Cicirello VA, Smith SF: Wasp-like Agents for Distributed Factory Coordination. Autonomous Agents and Multi-Agent Systems 2004;8:237-266.
22 Moon DH, Kim HS, Song C: A Simulation Study for Implementing Color Rescheduling Storage in an Automotive Factory. SIMULATION 2005;81:625-635.
23 Meissner S: Controlling just-in-sequence flow-production. Logistics Research 2010;2:45-53.
24 Jayaraman A, Narayanaswamy R, Gunal AK: A Sortation System Model; Simulation Conference, 1997., Proceedings of the 1997 Winter. 1997, pp 866-871.
25 Fournier X, Agard B: Improvement of earliness and lateness by postponement on an automotive production line. International Journal of Flexible Manufacturing Systems 2007;19:107-121.
26 Boysen N, Zenker M: A decomposition approach for the car resequencing problem with selectivity banks. Computers & Operations Research 2013;40:98-108.
27 Moon DH, Song C, Ha JH: A Dynamic Algorithm for the Control of Automotive Painted Body Storage. SIMULATION 2005;81:773-787.
28 Choi W, Shin H: A real-time sequence control system for the level production of the automobile assembly line. Computers & Industrial Engineering 1997;33:769-772.
29 Valero-Herrero M, Garcia-Sabater JP, Maheut J: An approach to the real circumstances of the car sequencing problem; 41st International Conference on Computers and Industrial Engineering. Los Angeles (USA), 2011.
30 Valero-Herrero M, Garcia-Sabater JP, Coronado-Hernandez JR, Maheut J: Planteamiento dinámico del problema de secuenciación en líneas de montaje con mezcla de modelos; XV Congreso de Ingeniería de Organización // 5th International Conference on Industrial Engineering and Industrial Management. Cartagena, 2011, pp 288-296.
31 Garcia-Sabater JP: The problem of JIT dynamic sequencing. A model and a parametric procedure; 1st Operations Research Post-graduate Peripatetic Programme ORP3 2001, Paris, September 26-29. 2001, vol Paris.
32 Monden Y: Toyota Prduction System: An integrated approach. Second Edition; Industrial Engineering and Management Press , 1993.

MISTA 2013

An Investigation of Multistage Approaches to Examination Timetabling

Syariza Abdul Rahman · Andrzej Bargiela · Edmund Burke · Barry McCollum · Ender Özcan

Abstract Many successful approaches to examination timetabling consist of multiple stages, in which a constructive approach is used for finding a good initial solution, and then one or more improvement approaches are employed successively to further enhance the quality of the solution obtained during the previous stage. Moreover, there is a growing number of studies describing the success of approaches which make use of multiple neighbourhood structures. In this study, we investigate the methods of ordering neighbourhood structures within a Variable Neighbourhood Search approach using a great deluge move acceptance method. We also analyse how this approach performs as an improvement algorithm when combined with different initialisation strategies while performing multiple runs for examination timetabling. The empirical results over a well known examination timetabling benchmark show that the performance of Variable Neighbourhood Search great deluge performs competitively, ranking second among previously proposed approaches, with the right choice of initialisation and neighbourhood ordering methods.

Syariza Abdul Rahman
Universiti Utara Malaysia, School of Quantitative Sciences, College of Art and Science, 06010 Sintok, Malaysia
E-mail: syariza@uum.edu.my

Andrzej Bargiela and Ender Özcan
University of Nottingham, School of Computer Science, Jubilee Campus,Nottingham NG8 1BB, UK
E-mail: {abb,exo}@cs.nott.ac.uk

Edmund Burke
University of Stirling, Cottrell Building, Stirling FK9 4LA, UK
E-mail: e.k.burke@stir.ac.uk

Barry McCollum
University of Queen Belfast, School of Electronics, Electrical Engineering and Computer Science, University Road, Belfast, BT7 1NN, Northern Ireland, UK
E-mail: b.mccollum@qub.ac.uk

1 Introduction

Examination timetabling is a well known computationally difficult real-world combinatorial optimisation problem ([32], [19]). Constructing a solution requires the assignment of a set of examinations to a limited number of time-slots subject to certain constraints which are frequently classified as hard and soft. The hard constraints must be satisfied for a solution to be considered *feasible*. A standard example of a hard constraint is the requirement that students cannot sit two examinations at the same time. Soft constraints, such as allowing students free time between examinations represent what are commonly referred to as preferences. An examination timetabling solution method aims to reduce the number of violations of such constraints while producing feasible high quality solutions.

As an indication of the inherent difficulty of examination timetabling, one of the largest problem instances solved to optimality as a real world problem Yeditepe University is reported in [34]. It has thirty-eight scheduled examinations. Due to the size of real world instances, it is therefore not surprising that the goal of examination timetabling is frequently defined as finding a high quality schedule for a given set of examinations subject to various institutional and individual preferences. The approaches to examination timetabling are mostly based on two search paradigms: construction and improvement [13]. A constructive approach starts with an empty solution and incrementally builds towards a complete solution. For example, graph colouring heuristics are commonly used as construction heuristics in examination timetabling. On the other hand, an improvement approach uses a complete or feasible solution and is concerned with further refining the quality of that initial solution. In the work presented here, we use an iterative framework which employs a set of mutational neighbourhood structures making random modifications and using local search.

Solution methods for examination timetabling problems have been widely investigated over the last decade or so. These methods include exact ([38]), constraint-based ([29]), heuristic ([5]), meta-heuristic ([14]), hyper-heuristic ([12], [22]) and multi-objective ([33]) approaches. Meta-heuristic techniques can be thought of as usually comprising two major classes of methodology i.e. stochastic local search-based techniques that deal with a single candidate solution at each iteration, and population-based techniques that maintain a set of candidate solutions during the search process. The stochastic local search-based techniques include tabu search ([23]), simulated annealing ([9]), iterated local search ([17]), large neighbourhood search ([1]) and variable neighbourhood search ([11]), whilst the population-based techniques include genetic algorithms ([18]), memetic algorithms ([22])and ant algorithm ([21]). Some other recent approaches applied to timetabling are hyper-heuristics ([15], [36]), case-based reasoning ([41]), fuzzy-based methods([7]), granular modeling ([4]), the developmental approach ([35]) and a harmony search algorithm ([6]). A review of the major approaches in examination timetabling can be found in [19] and [37].

Many successful approaches to examination timetabling, for example [17] and [31], consist of multiple stages, in which a constructive approach is used for finding a good quality initial solution, and then one or more improvement approaches are employed successively to further improve the quality of solution obtained during the previous stage. In most of the cases in which single point based search is performed, there are two main stages. An approach is used to build an initial solution which is fed into an improvement algorithm. For example, the winning approach in the 2007 International Timetabling Competition (ITC2007), uses constraint programming to build a solution

which is then passed to a stochastic local search algorithm for improvement ([31]). If a deterministic approach is used to construct an initial solution and the improvement algorithm is a stochastic algorithm, then for each trial of the improvement algorithm for a given instance, the same initial solution has to be used. If the initialisation approach employs randomness within, then the improvement approach can use a different initial solution produced by that approach at each trial. The main objective of this study is to investigate the influence of various initialisation strategies on a stochastic constructive approach which is used in the first stage and the influence on neighbourhood ordering methods on the performance of variable neighbourhood search (VNS) algorithm which is used in the succeeding stage. Within the later stage, VNS is combined with a great deluge acceptance method, allowing the acceptance of some worsening solutions for solving the examination timetabling problem. A range of methods managing different neighbourhood structures were tested as a part of this approach. The influence of initialization strategies within the construction phase and neighbourhood orderings within the VNS during the second phase are investigated in terms of the quality of resultant solutions across the Toronto benchmark instances ([20]). As part of further work, it is intended to further apply the technique to the examination datasets associated with the ITC2007 competition ([27])

The components of the VNS approach for examination timetabling are described in Section 2, with a focus on the algorithmic components, neighbourhood structures, neighbourhood ordering, initialisations and acceptance criteria. Section 3 provides the experimental results and remarks. Finally, the conclusion is presented in Section 4.

2 VNS for Examination Timetabling

It is a common problem that the the search process guided solely by a local search method could get trapped at a local optimum. The use of a different neighbourhood structure could potentially lead to a different local optimum ([30], [24]). Therefore, the use of multiple neighbourhood structures were introduced in [30] with the aim of ensuring that the search process is able to continue to explore the local solution space without becoming trapped and therefore stagnating. Variable neighbourhood search (VNS) performs an iterative search over candidate solutions from one neighbourhood structure to another whenever necessary. This algorithm can be considered as a single point-based selection hyper-heuristic [13] which manages multiple neighbourhood structures.

2.1 Neighbourhood Structures

It is known that the choice of neighbourhood structure influences the search process in VNS ([30], [24]). The purpose of employing more than one neighbourhood structure is to be able to combine the strengths of multiple operators during the search process dynamically. If one neighbourhood structure fails to improve the current solution, then the other neighbourhood structures might still have a chance for a better exploitation. Similarly, if the search gets stuck at a local optimum, the step size of the mutation operator in use could be small to overcome the trap, but another operator which supports a larger step size could provide the diversification needed to explore the space better.

Recently, Kempe-chain moves have been successfully employed in solving timetabling problems ([11], [3]). A standard neighbourhood structure as discussed in [39] is formed by a single move neighbourhood which consists of choosing an examination randomly and rescheduling it into a new feasible time-slot. In this study, along with this simple neighbourhood structure, Kempe-chain moves are introduced.

In a general implementation, the Kempe-chain move operates over two subsets of examinations by swapping between two feasible time-slots. Each subset of examinations is connected by edges to represent conflict between the examinations. In this study, the neighbourhood structures allow for infeasible moves. In the case of the Kempe-chain neighbourhood, the infeasible move may be due to more examinations from different time-slots being connected to the chosen examination. One of the aims of using multiple neighbourhood structures in VNS is to shake the current solution in various ways. In this case, the neighbourhood structures shake the current solution by allowing an infeasible move. In order to treat this infeasibility, a repair mechanism is used by the VNS algorithm.

Fifteen neighbourhood structures for examination timetabling are considered in this study. Traditionally, neighborhood structures are ordered in increasing step size used to perturb a candidate solution. The biased neighbourhood structures (referred to as number 8, 9, 11 and 12 below) were also used as described in [11] and [2] which choose the examination generating the highest penalty value for rescheduling from a number of randomly selected examinations. The aim is to repair an examination that contributes a large value to the cost function. Experiments using different sets of neighbourhood structures would also be useful. The implemented neighbourhood structures are ordered as follows by choosing:

1. One examination at random and move to a new random feasible time-slot.
2. Two examinations at random and move each examination to a new random feasible time-slot.
3. Two examinations at random and swap the time-slots between these two examinations. The feasibility of the two examinations is maintained.
4. Three examinations at random and move each examination to a new random feasible time-slot.
5. Four examinations at random and move each examination to a new random feasible time-slot.
6. Five examinations at random and move each examination to a new random feasible time-slot.
7. One move of Kempe-chain with one random examination.
8. One move of Kempe-chain with one examination selected randomly from 10% selection of examinations that give highest penalty.
9. One move of Kempe-chain with one examination selected randomly from 20% selection of examinations that give highest penalty.
10. Two moves of Kempe-chain with one random examination.
11. Two moves of Kempe-chain with one examination selected randomly from 10% selection of examinations that give highest penalty.
12. Two moves of Kempe-chain with one examination selected randomly from 20% selection of examinations that give highest penalty.
13. Two time-slots at random and swap between them.
14. One time-slot at random and move to a new feasible time-slot.
15. Shuffle all time-slots at random.

2.2 Ordering of Neighbourhood Structures

The studies by [24], [10], [8] and [2] have shown that the ordering of neighbourhood structures within the VNS framework could considerably influence the solution quality obtained at the end. The neighbourhood structures are frequently ordered on the basis of a pre-defined sequence with increasing step size, to achieve a better solution quality at the end of a run. [10], Burke et al. introduced a parameter $success_k$ to penalise a non-improving neighbourhood structure. In the case that an improvement in the solution quality is detected, search always starts using the first neighbourhood with the smallest size. Any neighbourhood that has $success_k$ value which is less than 1, cannot have priority in the next iteration, since this indicates that the neighbourhood cannot improve the solution quality. In another study, [8] observed that a combination of three different neighbourhood structures yielded an improved performance. Nevertheless, a recent study by [11] showed that fixing the ordering of neighbourhood structures was not essential. The study demonstrated that VNS and a hybridisation with a genetic algorithm could produce a good quality solution. Since the solution quality was dependent on the selection of the neighbourhood, the genetic algorithm worked intelligently by selecting a neighbourhood structure from the VNS framework.

Motivated from these studies, an investigation has been undertaken into the ordering of neighbourhood structures, in which five variants of neighbourhood ordering strategies are explored. The first and second variants are the ordering of neighbourhood structures based on an increasing step size, as proposed in [10] and [2], respectively. The first variant is a basic VNS where the neighbourhood structure starts with $k = 1$. The solution search continues with the next neighbourhood k if the improvement to the solution quality can not be found. Once an improvement to the solution quality is found, the search starts back with $k = 1$; the second variant is related with the neighbourhood structure used in the previous iteration. In this variant, a neighbourhood k that has been used to improve the solution quality in the previous iteration is used in the next iteration for finding good solution quality. These variants of neighbourhood orderings are represented as 'basic VNS' and 'start-k', respectively.

The third variant of neighbourhood ordering is based on a strategy adapted from squeaky wheel optimisation [25], by assigning and increasing the penalty under a parameter (called 'priority') of a neigbourhood structure yielding a worsening move. If there is no improvement in the solution quality while using the current neighbourhood, then the neighbourhood with lowest priority is chosen next. This ordering strategy is denoted as 'adaptive I'. A variant of 'adaptive I' considers the effect of neighbourhood structures using a small step size which is a faster operation than the third ordering strategy, but this gives relatively greater priority to neighbourhood structures using a small step size, which is referred to as 'adaptive II'. In the case of ties, where more than one neighbourhood has the same priority value, then the neighbourhood structure with the smallest step size is chosen in the next iteration.

The neighbourhood structures are first classified with respect to the step size used in a move, as small and large. For example, a neighbourhood operator with a small step size consists of a single move of an examination to a new feasible time-slot, whilst a neighbourhood operator with a large step size consists of a number of examinations to be moved and might incur large differences in the solution quality when the chosen move takes place, for instance, when allocations in two time-slots are swapped at random. This neighbourhood structure involves all examinations in one time-slot to be swapped with all examinations in the other time-slot.

As the fifth variant of neighbourhood ordering, a reinforcement learning mechanism is also investigated. Reinforcement learning ('RL') is a mechanism that interacts with the behaviour of an environment by assigning punishments and rewards based on its performance. In this case, a neighbourhood operator is given a reward if there is improvement in the solution quality after its application, while a punishment is given if the move fails to improve the current solution.

2.3 Initialisation Strategies

This study is concerned with improving the initial solution obtained from the constructive approaches presented in a previous study. The main objective is to understand the influence of different kinds of initialisation strategies on the solution quality when an improvement approach is employed in searching the landscape starting from that initial solution which is locally optimal. Several studies showed that good initial solutions can ultimately translate to a further improvement in the solution quality within a short time ([16]). Starting from a different random point (solution) in the search space at each trial frequently helps, particularly if a stochastic local search algorithm is used for solving a given problem. In many cases, the initial solutions are extremely poor. Recent studies indicate the success of multi-stage hybrid algorithms in which an already improved solution is constructed and then it is further improved by another algorithm. While testing a two-stage stochastic local search algorithm, certain choices can be made. The algorithm dealing with the second stage will not receive a totally random initial solution and will be improving upon the solution generated in the first stage which will potentially be a locally optimal solution. Three different variants of initial solutions are considered to be used in the second stage of our algorithm, i.e. *poor*, *good* and *multiple*. The good initial solution is the best one obtained using the previously proposed constructive approaches, while a poor initial solution has a quality level which is at least 20% worse than the solution quality of that good initial solution. During the multiple trials of our stochastic local search algorithm for solving a given instance, good and poor initial solutions are used separately as an initial solution during all runs, while multiple initial solutions are used individually at each trial. It is possible that some of the multiple solutions are not feasible and can be dealt with during the improvement phase with a repair mechanism that is incorporated into the VNS-GD algorithm. The initial solutions are obtained from the constructive approach of Adaptive Heuristic Orderings described in [5].

2.4 Acceptance Criteria of VNS

Algorithm 1 illustrates the pseudo-code of the VNS algorithm using the great deluge algorithm for acceptance criteria. The great deluge algorithm is a local search approach that accepts a worsening solution based on an acceptance level of quality, denoted as B which is decreased with a certain amount called the decay rate, denoted as β. This approach has demonstrated a good performance for timetabling ([28], [40]). Variants of the decay rate have been investigated by [40], [26] and [28].

The decay rate, β for this study is set as 0.001. This value is chosen as a result of some initial tests which are not reported in this paper due to space restrictions. B is initialised using the initial solution quality. In Algorithm 1, let s be an initial solution

and is set as the best solution, s_{best}, obtained so far. The quality of solution s, $f(s)$ is set as $f(s_{best})$. While the algorithm starts the search, a solution s' is generated randomly by visiting the k^{th} neighbourhood sequentially until a local optimum s'' is found. The solution s'' is accepted whenever the solution quality of s'', $f(s'')$ is better than $f(s_{best})$. Otherwise, if $f(s'')$ is better than B, then the solution s'' is accepted. Afterwards, B is reduced proportional to β. Every time the solution quality, $f(s)$ is accepted, the search will continue with the next k that is identified based on an ordering strategy of neighbourhood structure described in Section 2.2.

Algorithm 1 Variable Neighbourhood Search - Great Deluge

Select the set of neighbourhood structures $N_k, k = 1, ..., k_{max}$, to be used in the search; choose stopping condition;
set initial solution s; $s_{best} \leftarrow s$; $f(s_{best} \leftarrow f(s))$;
Estimate the acceptance level of quality to be accepted, $B = f(s)$; set the decay rate, β;
Repeat until stopping criteria is satisfied:
1. Set $k := 1$;
2. Until $k = k_{max}$, repeat:
(a) Shaking: Generate a point s' at random from the k^{th} neighbourhood of $s(s' \in N_k(s))$;
Repair mechanism:
(b) Local Search: Apply a local search method with s' as initial solution until local optimum s'' is obtained.
(c) Move or not:
Calculate $f(s'')$
Great deluge acceptance criteria:
if $f(s'')$ is better than $f(s_{best})$ **then**
  $s \leftarrow s''$
  $s_{best} \leftarrow s''$
else
  if $f(s'')$ is better than B **then**
    $s \leftarrow s''$
  end if
end if
$B = B - \beta$
Continue the search with identified k.

The proposed approach is incorporated with a repair mechanism since infeasible moves can happen. The pseudo-code of the repair mechanism for the Toronto benchmark datasets is shown in Algorithm 2. Delta evaluation is always applied for fast execution of the proposed algorithm. An examination is checked against all possible values of delta_cost obtained by an assignment to each possible time-slot. Once the lowest delta_cost is found, the examination, i_{best} is moved to that best time-slot, j_{best}, and the process continues until there is no improvement in the solution quality.

3 Experiments and Results

The experiments in this study are performed using the examination timetabling datasets from various universities, introduced by [20][1]. The version I of the datasets as introduced in [37] is employed as a test bed for the proposed approaches. The objective for the Toronto benchmark problem is to create a feasible timetable so that no student is

[1] It can be accessed at ftp://ftp.mie.utoronto.ca/pub/carter/testprob/

Algorithm 2 Repair mechanism for the Toronto benchmark datasets

```
Repair mechanism:
while the delta_cost > 0 do
  for i = 1 to number of examinations do
    for j = 1 to number of slots do
      Find the lowest delta_cost;
      i_best ← i;
      j_best ← j;
    end for
  end for
  Move i_best to j_best
end while
```

required to sit two examinations at any one time. To achieve a high quality timetable, the soft constraints need to be satisfied as much as possible. Thus, during the timetable construction, it is required that student's examinations are assigned as far apart as possible. The proximity cost function introduced in [20] in conjunction with the introduced datasets was used in order to measure the quality of the obtained timetable and to describe the average penalty of students distributed in the examination schedule.

Pentium IV 1.86 GHz. Windows machines having 1.97 Gb memory were used during the experiments. The stopping conditions for the experiments were set as 50000 iterations. However, the running time for the initial solutions is not included during the improvement phase because of the challenge of comparing the performance of the algorithm using different initial solution values. 100 runs were obtained for each dataset tested with three different initial solutions and with different types of neighbourhood orderings. The results are provided in the tables below, each table representing the results for each initial solution tested with different neighbourhood orderings. The best solution for each dataset is represented in bold font.

Tables 1, 2 and 3 illustrate the results of poor, good and multiple initial solutions respectively, tested with different neighbourhood orderings for the Toronto benchmark datasets. The initial values of the poor and good solutions for each of the datasets are also provided in the tables. It is intended that these values will be improved using the VNS-GD algorithm, repeating each value for a hundred runs. However, since the multiple initial solution is generated at each run before proceeding to the improvement phase, only the average value of all generated initial solutions is provided.

The overall results with poor initial solution are presented in Table 1. They show that the basic VNS that always starts with $k = 1$ whenever there is improvement to the solution quality, generates most of the best solutions with four best results and two ties for kfu93 and sta83 I. With other neighbourhood orderings, start-k obtained four best results with one tie, adaptive I obtained one best result and one tie with other types of neighbourhood orderings, adaptive II obtained one best result but tied with basic VNS. RL obtained one best result and a tie for sta83 I. The standard deviation (of less than one) for different neighbourhood orderings is relatively small.

The results generated using the good initial solutions indicate the same pattern as for the poor initial solutions, where the basic VNS obtained the best results for six problems and a tie for sta83 I. On the other hand, the other neighbourhood ordering types obtained three best results for start-k and one best result for the adaptive II and RL. The rest of the neighbourhood orderings are tied for sta83 I. Considering the standard deviation value, the good initial solution provides less than the poor initial solution. The poor initial solution contributes a higher standard deviation as expected

Table 1 The results of poor initial solutions tested with different neighbourhood orderings for the Toronto benchmark datasets (RL = reinforcement learning, stdev. = standard deviation, Av. t(m) = average running time in minutes)

		One poor initial solution						
Problem	Initial value	Start-k	Basic VNS	Adaptive I	Adaptive II	RL	Av. t(m)	Best
car91	6.35	4.95	**4.89**	4.97	4.97	4.97	518.95	4.89
stdev.		0.06	0.06	0.06	0.07	0.07		
car92	5.43	4.19	4.18	**4.14**	4.15	4.18	274.76	4.14
stdev.		0.07	0.06	0.06	0.05	0.05		
ear83 I	47.85	**33.45**	33.55	33.52	33.63	33.60	18.56	33.45
stdev.		0.52	0.59	0.51	0.51	0.55		
hec92 I	13.91	10.29	**10.21**	10.25	10.38	10.33	1.33	10.21
stdev.		0.15	0.14	0.14	0.14	0.14		
kfu93	18.03	13.48	**13.46**	13.56	13.46	13.52	68.87	13.46
stdev.		0.15	0.15	0.15	0.15	0.14		
lse91	14.29	**10.53**	10.55	10.60	10.57	10.55	57.30	10.53
stdev.		0.19	0.19	0.15	0.17	0.19		
rye93	11.71	**8.37**	8.45	8.57	8.50	8.61	120.96	8.37
stdev.		0.10	0.10	0.10	0.11	0.11		
sta83 I	196.68	**157.06**	**157.06**	**157.06**	157.07	**157.06**	1.06	157.06
stdev.		0.06	0.05	0.06	0.06	0.05		
uta92 I	4.40	3.41	**3.40**	3.44	3.43	3.42	485.08	3.40
stdev.		0.05	0.04	0.03	0.04	0.04		
ute92	32.80	25.04	**24.96**	25.08	25.06	24.96	4.06	24.96
stdev.		0.10	0.10	0.11	0.11	0.10		
tre92	10.91	**8.18**	8.26	8.32	8.28	8.29	38.08	8.18
stdev.		0.11	0.13	0.09	0.10	0.10		
yor83 I	50.48	36.22	36.77	36.23	36.71	**36.05**	10.32	36.05
stdev.		0.40	0.40	0.39	0.36	0.44		

and this is due to the poor starting-point which tends to generate a variation in the final solution quality.

The results from the multiple initial solutions also appear to work well with this approach, indicating the same pattern when the solutions are generated. The basic VNS shows great success with the best solutions for six problems, while start-k, adaptive I and RL, each obtained one best result respectively while adaptive II obtained two best results. The rest of the neighbourhood orderings are tied for the sta83 I problem. The standard deviation among the neighbourhood orderings is clearly about the same, but there are some significant differences when different types of initialisation strategies are compared.

Among the problems, sta83 I shows almost no differences when employed with different types of neighbourhood ordering. Even when sta83 I is addressed with different initial solutions, the move is tended to get stuck at local optimum and no further improvement could be obtained. This may be due to the incorporation of the repair mechanism that checked and moved each examination and time-slot which could reduce the least penalty cost to the lowest level. Although there are variations in the standard deviation values for different initial solutions used for the sta83 I problem, the start with good initial solution shows that the standard deviations (less than 0.004) are very small.

The overall results demonstrate that the basic VNS-GD performed very well where it obtained most best results over the thirteen problems of the Toronto benchmark

Table 2 The results of good initial solutions tested with different neighbourhood orderings for the Toronto benchmark datasets (RL = reinforcement learning, stdev. = standard deviation, Av. t(m) = average running time in minutes)

		One good initial solution						
Problem	Initial value	Start-k	Basic VNS	Adaptive I	Adaptive II	RL	Av. t(m)	Best
car91	5.08	**4.86**	4.87	4.89	4.94	4.88	542.65	4.86
stdev.		0.03	0.03	0.03	0.02	0.03		
car92	4.34	4.24	4.22	4.22	4.21	**4.20**	286.64	4.20
stdev.		0.02	0.01	0.01	0.01	0.02		
ear83 I	36.91	34.36	33.81	34.08	**33.76**	34.17	17.62	33.76
stdev.		0.54	1.01	0.69	0.69	0.63		
hec92 I	11.13	**10.11**	10.19	10.28	10.23	10.30	1.38	10.11
stdev.		0.15	0.15	0.14	0.16	0.14		
kfu93	14.42	13.83	**13.62**	13.89	13.80	13.84	60.38	13.62
stdev.		0.06	0.09	0.05	0.07	0.06		
lse91	11.41	10.65	**10.58**	10.65	10.66	10.59	54.89	10.58
stdev.		0.12	0.13	0.14	0.12	0.11		
rye93	9.37	8.51	**8.45**	8.51	8.50	8.53	133.74	8.45
stdev.		0.08	0.08	0.08	0.09	0.08		
sta83 I	157.34	**157.08**	**157.08**	**157.08**	**157.08**	**157.08**	1.10	157.08
stdev.		0.00	0.00	0.00	0.00	0.00		
uta92 I	3.52	3.49	**3.46**	3.48	3.48	3.47	496.32	3.46
stdev.		0.00	0.01	0.01	0.01	0.01		
ute92	26.24	**24.95**	24.99	24.97	25.04	24.97	4.52	24.95
stdev.		0.12	0.13	0.11	0.11	0.13		
tre92	8.73	8.36	**8.28**	8.31	8.37	8.38	36.08	8.28
stdev.		0.06	0.08	0.06	0.06	0.06		
yor83 I	39.67	37.53	**36.51**	37.14	37.19	37.31	10.39	36.51
stdev.		0.35	0.52	0.42	0.38	0.39		

datasets when tested with various initialisations. This suggests that the neighbourhood ordering with respect to the size of the neighbourhood can affect the search for good solutions. On each occasion when there is improvement to the solution quality, the search always starts with a small neighbourhood structure. This would allow the search to explore more regions that cannot be achieved by other larger neighbourhood structures, while at the same time it could reduce the processing time because the search always begins with a small size neighbourhood structure.

The running time for this approach is quite long because of the incorporation of a repair mechanism. The repair mechanism for the Toronto benchmark datasets considers each examination to be repaired or improved, taking into consideration a move to time-slot that can reduce the current penalty cost to the lowest level. In this study, the repair mechanism not only works for the infeasible moves but it also tries to repair each examination assignment by reducing the assignment cost.

Any VNS-based approach initialised with the multiple solutions which are all obtained by the Adaptive Heuristic Orderings approach ([5]) performs the best based on the best-of-runs results across almost all instances. Moreover, initialising VNS-based approaches using a single solution during all trials regardless whether that solution is good or poor does not perform as well.

Tables 4 (a) and (b) illustrate the comparison of the results from different improvement approaches in the literature with our VNS-GD approach tested with multiple initial solution and basic VNS ordering strategies. In order to see the best approach,

Table 3 The results of multiple initial solutions tested with different neighbourhood orderings for the Toronto benchmark datasets (RL = reinforcement learning, stdev. = standard deviation, Av. t(m) = average running time in minutes)

	Av.	Multiple initial solution						
Problem	initial value	Start-k	Basic VNS	Adaptive I	Adaptive II	RL	Av. t(m)	Best
car91	5.65	4.87	4.88	4.89	4.87	**4.83**	519.42	4.83
stdev.		0.08	0.07	0.08	0.07	0.08		
car92	4.93	4.10	**4.06**	4.07	4.13	4.12	299.9	4.06
stdev.		0.06	0.05	0.06	0.18	0.06		
ear83 I	41.94	33.43	**33.22**	33.38	33.53	33.41	19.49	33.22
stdev.		0.52	0.70	0.47	0.47	0.63		
hec92 I	12.76	10.25	10.27	10.28	**10.23**	10.35	1.35	10.23
stdev.		0.16	0.17	0.16	0.14	0.15		
kfu93	16.23	13.39	**13.30**	13.57	13.46	13.41	66.36	13.30
stdev.		0.18	0.17	0.13	0.15	0.16		
lse91	12.82	10.45	10.45	**10.36**	10.38	10.46	56.81	10.36
stdev.		0.25	0.27	0.26	0.25	0.28		
rye93	11.08	8.56	**8.42**	8.53	8.48	8.53	140.12	8.42
stdev.		0.09	0.11	0.13	0.10	0.12		
sta83 I	160.16	**157.04**	**157.04**	**157.04**	**157.04**	**157.04**	1.37	157.04
stdev.		0.08	0.08	0.07	0.07	0.07		
uta92 I	3.88	**3.36**	3.38	3.37	**3.36**	3.40	504.24	3.36
stdev.		0.04	0.05	0.05	0.05	0.05		
ute92	28.55	24.97	**24.92**	24.99	24.93	24.93	4.38	24.92
stdev.		0.12	0.13	0.09	0.09	0.09		
tre92	9.58	8.25	**8.12**	8.22	8.24	8.22	39.38	8.12
stdev.		0.11	0.13	0.11	0.11	0.14		
yor83 I	45.18	36.48	**35.88**	36.27	36.37	35.96	10.69	35.88
stdev.		0.35	0.50	0.42	0.40	0.38		

the results of each problem are ranked and the best approach is identified based on the least average rank. The rank value of each approach is provided in brackets next to the solution quality in each table. Based on the average ranking of approaches across all instances considering their best performances, the VNS-GD approach is placed as the second best approach. However, it did not obtain the best result for any of the benchmark problems. The best approach is the study provided by [11] which employs the VNS approach hybridized with a genetic algorithm. The results of pur93 I were not included in the previous tables since it required a long running time and it was almost impossible to obtain the results for a hundred runs due to the size of the problem. The run for pur93 I was performed with only a good initial solution starting with solution quality (5.74), and was repeated only three times. The best result for pur93 I is presented in Table 4.

4 Conclusion

This study investigated different initialisation strategies in order to perform multiple runs using a variable neighbourhood search algorithm hybridised with a great deluge move acceptance method (VNS-GD). The overall framework represents a two-stage stochastic search algorithm which uses the VNS-GD in the second stage. We have simulated the behavior of a deterministic algorithm in the first stage by using a single

Table 4 Comparison of different improvement approaches with VNS-GD. The rank of each approach as compared to the rest for each instance is indicated in parenthesis.

Problem	[23]	[33]	[16]	[29]	[41]	[18]
car91	6.2 (10)	-	4.65 (3)	5.1 (6)	**4.5** (1)	5.4 (9)
car92	5.2 (10)	-	4.1 (4.5)	4.3 (7.5)	3.93 (2)	4.2 (6)
ears83 I	45.7 (12)	38.9 (11)	37.05 (10)	35.1 (8)	33.71 (4)	34.2 (5)
hec92 I	12.4 (12)	11.2 (10)	11.54 (11)	10.6 (6)	10.83 (8)	10.4 (5)
kfu93	18 (12)	16.5 (11)	13.9 (8)	13.5 (4.5)	13.82 (7)	14.3 (9)
lse91	15.5 (12)	13.2 (11)	10.82 (8)	10.5 (7)	10.35 (5)	11.3 (9.5)
pur93 I	-	-	-	-	-	-
rye92	-	-	-	8.4 (2)	8.53 (4)	8.8 (6)
sta83 I	160.8 (11)	158.1 (6)	168.73 (12)	157.3 (4.5)	158.35 (9)	157 (2)
tre92	10 (12)	9.3 (10)	8.35 (5)	8.4 (6.5)	7.92 (3)	8.6 (8.5)
uta92 I	4.2 (11)	-	3.2 (3.5)	3.5 (8)	**3.14** (1)	3.2 (3.5)
ute92	29 (12)	27.8 (11)	25.83 (7)	25.1 (4)	25.39 (6)	25.3 (5)
yor83 I	41 (12)	38.9 (10)	37.28 (8)	37.4 (9)	36.53 (7)	36.4 (6)
Av. Rank	11.15	10.38	7.62	6.35	5.04	6.38
Rank	12	11	9	6	5	7

(a)

Problem	[1]	[21]	[17]	[11]	[40]	VNS-GD
car91	5.2 (7)	5.2 (7)	6.6 (11)	4.6 (2)	4.8 (4)	4.88 (5)
car92	4.4 (9)	4.3 (7.5)	6 (11)	**3.9** (1)	4.1 (4.5)	4.06 (3)
ears83 I	34.9 (6)	36.8 (9)	**29.3** (1)	32.8 (2)	34.92 (7)	33.22 (3)
hec92 I	10.3 (4)	11.1 (9)	**9.2** (1)	10 (2)	10.73 (7)	10.27 (3)
kfu93	13.5 (4.5)	14.5 (10)	13.8 (6)	**13.0** (1.5)	**13.0** (1.5)	13.3 (3)
lse91	10.2 (4)	11.3 (9.5)	**9.6** (1)	10 (2)	10.01 (3)	10.45 (6)
pur93 I	-	4.6 (2)	**3.7** (1)	-	4.73 (3)	5.71 (4)
rye92	8.7 (5)	9.8 (8)	**6.8** (1)	-	9.65 (7)	8.42 (3)
sta83 I	159.2 (10)	157.3 (4.5)	158.2 (7)	**156.9** (1)	158.26 (8)	157.04 (3)
tre92	8.4 (6.5)	8.6 (8.5)	9.4 (11)	7.9 (2)	**7.88** (1)	8.12 (4)
uta92 I	3.6 (10)	3.5 (8)	3.5 (8)	3.2 (3.5)	3.2 (3.5)	3.38 (6)
ute92	26 (8)	26.4 (10)	**24.4** (1)	24.8 (2)	26.11 (9)	24.92 (3)
yor83 I	36.2 (3.5)	39.3 (11)	36.2 (3.5)	**34.9** (1)	36.22 (5)	35.88 (2)
Av. Rank	6.62	8.00	4.88	3.00	4.88	3.69
Rank	8	10	3.5	1	3.5	2

(b)

poor solution and a single good solution obtained from previously proposed approaches for each instance. The same poor and good solutions are used separately at each run of the proposed algorithm to see their influence on the performance of the overall algorithm. Also, we used a previously proposed stochastic constructive algorithm to build multiple solutions for each run of VNS-GD. Moreover, the performance of the overall framework combining different neighbourhood ordering methods with different initialisation strategies is tested. Various initialisations and neighbourhood ordering influenced the resultant solution quality as expected. The use of different initial solutions at each run, even though these solutions are already locally optimum, still demonstrates good performance for examination timetabling. Considering the neighbourhood orderings, the overall framework performed the best with a strategy which grows the step size whenever there is no improvement on the Toronto benchmark. Additionally, our approach ranks the second considering all instances based on the best of runs for each

instance when compared to previously proposed approaches. The intention is to fully analyse and understand the results presented here before applying the method to, in the first instance, the ITC2007 examination benchmark datasets. Although the results so far are extremely encouraging, it is important to carry out further evaluation and testing.

References

1. Abdullah, S., Ahmadi, S., Burke, E. K., & Dror, M., Investigating Ahuja-Orlin's large neighbourhood search approach for examination timetabling, OR Spectrum, 29(2), 351-372 (2007)
2. Abdullah, S., Burke, E. K. & McCollum, B., An investigation of variable neighbourhood search for university course timetabling, Proceedings of the 2nd Multi-disciplinary International Conference on Scheduling: Theory and Applications (MISTA), New York, USA, 18-21 July, 413-427. (2005)
3. Abdullah, S., Shaker, K., McCollum, B. & McMullan, P., Incorporating great deluge with Kempe chain neighbourhood structure for the enrolment-based course timetabling problem, Yu, J., Greco, S., Lingras, P., Wang, G. & Skowron, A. (Eds.), Rough Set and Knowledge Technology, Springer Berlin / Heidelberg, 6401, 70-77. (2010)
4. Abdul Rahim, S. K., Bargiela, A., & Qu, R., Granular modelling of exam to slot allocation, In roceedings of the 23rd european conference on modelling and simulation (ECMS), Madrid, Spain, 861-866. (2009)
5. Abdul Rahman, S., Bargiela, A., Burke, E. K., McCollum, B., & Özcan, E., Construction of examination timetables based on ordering heuristics, In proceedings of the 24th international symposium on computer and information sciences, 727-732. (2009)
6. Al-Betar, M. A., Khader, A. T. & Thomas, J. J., A combination of metaheuristic components based on harmony search for the uncapacitated examination timetabling, Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling, (PATAT 2010), 10-13 August 2010, Belfast, Northern Ireland, (2010)
7. Asmuni, H., Burke, E. K., Garibaldi, J. M., McCollum, B., & Parkes, A. J., An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables, Computers and Operations Research, 36(4), 981-1001 (2009)
8. Bolaji, A. L., Khader, A. T., Al-Betar, M. A., Awadallah, M. A. & Thomas, J. J., The effect of neighborhood structures on examination timetabling with artificial bee colony, Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling, (PATAT 2012), 29-31 August 2012, Son, Norway, (2012)
9. Burke, E. K., Bykov, Y., Newall, J. P., & Petrovic, S., A time-predefined local search approach to exam timetabling problem, IIE Transactions, 36(6), 509-528 (2004)
10. Burke, E. K., De Causmaecker, P., Petrovic, S. & Vanden Berghe, G., Metaheuristics: Computer decision-making, Chapter 7: Variable neighbourhood search for nurse rostering problems, Resende, M. G. C. & de Sousa, J. P. (Eds.) Kluwer, 153-172. (2003)
11. Burke, E. K., Eckersley, A. J., McCollum, B., Petrovic, S., & Qu, R., Hybrid variable neighbourhood approaches to university exam timetabling, European Journal of Operational Research, 206(1), 46-53 (2010)
12. Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P. & Schulenburg, S., Handbook of meta-heuristics Hyper-heuristics: An emerging direction in modern search technology, Glover, F. & Kochenberger, G. (Eds.) Kluwer, 457-474. (2003)
13. Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., & Özcan, E., A classification of hyper-heuristic approaches, In M. Gendreau and J-Y Potvin (Eds.), Handbook of metaheuristics, 449-468. Springer (2010)
14. Burke, E. K. & Kendall, G. (Eds.) Search methodologies: Introductory tutorials in optimization and decision support techniques, Springer, (2005)
15. 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)
16. Burke, E. K. & Newall, J. P., Enhancing timetable solutions with local search methods, Lecture notes in computer science: Practice and theory of automated timetabling IV: selected papers from the 4th international conference, Springer-Verlag, 2740, 195-206. (2003)

17. Caramia, M., Dell'Olmo, P, & Italiano, G. F., Novel local search based approaches to university examination timetabling, INFORMS Journal of Computing, 20, 86-99 (2008)
18. Côté, P., Wong, T. & Sabourin, R., A hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem, Lecture Notes in Computer Science: Selected papers from the 5th International Conference on Practice and Theory of Automated Timetabling (PATAT 2004), 3616, 151-168. (2005)
19. Carter, M. W. & Laporte, G., Recent developments in practical examination timetabling, Selected papers from the first international conference on practice and theory of automated timetabling, 3-21. Springer-Verlag (1996)
20. Carter, M. W., Laporte, G. & Lee, S., Examination timetabling: Algorithmic strategies and applications, Journal of the Operational Research Society, 47, 373-383 (1996)
21. Eley, M., Ant algorithms for the exam timetabling problem, Burke, E. K. & Rudova, H. (Eds.), Lecture notes in computer science: Practice and theory of automated timetabling VI: selected papers from the 6th international conference, Berlin: Springer, 3867, 364-382. (2007)
22. Ersoy, E., Özcan, E., & Sima Uyar, A., Memetic algorithms and hyperhill-climbers, In P. Baptiste, G. Kendall, A. M. Kordon, & F. Sourd (Eds.), Lecture notes in computer science: Multidisciplinary international conference on scheduling: theory and applications: selected papers from the 3rd international conference, 159-166. (2007)
23. Di Gaspero, L. & Schaerf, A., Tabu search techniques for examination timetabling, Lecture notes in computer science: Practice and theory of automated timetabling III: selected papers from the 3rd international conference, Berlin: Springer, 104-117. (2001)
24. Hansen, P. & Mladenović, N., Variable neighborhood search: Principles and applications, European Journal of Operational Research, 130, 449-467 (2001)
25. Joslin, D. E., & Clements, D. P., "Squeaky wheel" optimization, Journal of Artificial Intelligence Research, 10, 353-37 (1999)
26. Landa-Silva, D. & Obit, J. H., Great deluge with nonlinear decay rate for solving course timetabling problems, Proceedings of the 2008 IEEE Conference on Intelligent Systems (IS 2008), IEEE press, 11-18. (2008)
27. McCollum, B., McMullan, P., Burke, E. K., Parkes, A. J., & Qu, R. (2008). A new model for automated examination timetabling. Annals of Operations Research, 2, 2-3.
28. McCollum, B., McMullan, P., Parkes, A. J., Burke, E. K., & Abdullah, S., An extended great deluge approach to the examination timetabling problem, In proceedings of the 4th multidisciplinary international conference on scheduling: theory and applications (MISTA09), Dublin 424-434. (2009)
29. Merlot, L. T. G., Boland, N., Hughes, B. D. & Stuckey, P. J., A hybrid algorithm for the examination timetabling problem, Burke, E. K. & Erben, W. (Eds.), Lecture notes in computer science: Practice and theory of automated timetabling V: selected papers from the 4th international conference, Berlin: Springer, 2740, 207-231. (2003)
30. Mladenović, N. & Hansen, P., Variable neighborhood search, Computers & Operations Research, 24, 1097-1100 (1997)
31. Müller, T., ITC2007 solver description: a hybrid approach Annals OR, 172, 429-446 (2009)
32. Schindl, D., Some new hereditary classes where graph coloring remains NP-hard, Discrete Mathematics, 295 (1-3), 197-202 (2005)
33. Paquete, L. F. & Fortseca, C. M., A study of examination timetabling with multiobjective evolutionary algorithms, Proceedings of the 4th Metaheuristics International Conference (MIC 2001), 149-154. (2001)
34. Parkes, A. & Özcan, E., Properties of Yeditepe examination timetabling benchmark instances, Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling, 531-534. (2010)
35. Pillay, N., The revised developmental approach to the uncapacitated examination timetabling problem, Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, 187-192. (2009)
36. Pillay, N., Hyper-heuristics for educational timetabling, Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling, 316-340. (2012)
37. Qu, R., Burke, E. K., Mccollum, B., Merlot, L. T., & Lee, S. Y., A survey of search methodologies and automated system development for examination timetabling, Journal of Scheduling, 12(1), 55-89 (2009)
38. Qu, R., He, F. & Burke, E. K., Hybridizing integer programming models with an adaptive decomposition approach for exam timetabling problems, Proceedings of the 4th Multidisciplinary International Scheduling: Theory and Applications 2009, 435-446. (2009)

39. Thompson, J. & Dowsland, K., Variants of simulated annealing for the examination timetabling problem, Annals of Operational Research, 63, 105-128 (1996)
40. Turabieh, H., & Abdullah, S., An integrated hybrid approach to the examination timetabling problem, Omega, 39, 598-607 (2011)
41. Yang, Y. & Petrovic, S., A novel similarity measure for heuristic selection in examination timetabling, Proccedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004), (2004)

MISTA 2013

Tackling University Timetabling Problems Using Hybridization of Swarm Based Approach

Fong Cheng Weng • Hishammuddin bin Asmuni • Barry McCollum

1 Introduction

Real World Timetabling problems are usually complex NP hard optimization problems. The aim is to allocate a set of events or activities into a set of times and resources while satisfying a set of side requirements (or constraints). Examples of timetabling problems are transport scheduling, job shop scheduling, personnel rostering, machine scheduling and university timetabling. University timetabling is a challenging and repetitive task faced by educational institutions worldwide and is usually categorized into examination and course timetabling. Both of these areas are similar though there exists significant differences. The main difference is in relation to the time slot assignment part of the process i.e. multiple exams can be allocated into one time slot while only one course can be assigned into one time slot at one time. In addition, it is common practice that multiple exams can be assigned to one room which is not usually the case with course timetabling.

In both areas, two distinct types of constraints are taking into account during the timetabling construction: these can be classified as hard and soft constraints. Satisfaction of hard constraints is mandatory in order to generate a clash free timetable (known as feasible timetable) e.g. no students are required to attend two events at the same time. In relation to soft constraints, violation are acceptable but should be minimized as they affect the quality of the overall timetable. An example of a soft constraint is an attempt to provide students with the most time between two examinations that they are required to sit.

Fong Cheng Weng
Software Engineering Research Group, Software Engineering Department, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia
E-mail: chengweng0410@hotmail.com

Hishammuddin bin Asmuni
Software Engineering Research Group, Software Engineering Department, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia
E-mail: hishamudin@utm.my

Barry McCollum
Department of Computer Science, Queen's University Belfast, Belfast BT7 1NN United Kingdom
E-mail: b.mccollum@qub.ac.uk

Various automated approaches have been introduced over the last two decades in tackling university timetabling problems (Overview of research approaches can be seen at [3-4]). At early stage, the automated approaches were based on sequential heuristic mainly inspired from the theory of graph coloring. Subsequently, research into meta-heuristic approaches have prevailed within the research area due to improved overall performance. Generally, meta-heuristic approaches can be divided into two types: single-solution based and population based approaches. In single solution based approaches, changes to a single solution is investigated throughout the solution searching process. During the search, the solution (current solution) undergoes a sequence of changes governed by the meta-heuristic technique until a local optimal point (final solution) is reached i.e. the governing technique is capable of locally fine-tuning the solution search region(strong exploitation power). The key factors associated with single-solution based approaches are related to the solution acceptance rule definition of neighbourhood structures which ensures the connectivity of the search region. However, all single-solution based, in general, ultimately suffer from becoming trapped in local optima (weak exploration power).

On the other hand, population based approaches are capable of manipulating multiple solutions simultaneously during solution searching process. As compared with single-solution search approaches, population based approaches in general are good at exploring the problem search region as they able to explore multiple search regions (good exploration power). However, the tradeoff is they are weak in fine-tuning and investigating the promising search regions (weak exploitation power). It is clear that in order to provide a good solution for addressing university timetabling problems, the designed algorithm should be able to explore and exploit the solution search region effectively. Therefore, it is proposed a balance is required between the population and single solution meta-heuristic approaches

Hybridization of meta-heuristic approaches in addressing university timetabling problems has been investigated by an increasing number of researchers in recent years. The main purpose of such hybridization is to combine the strength of different approaches (particularly single-solution based with population based approaches) which are able to explore and exploit the associated solution search region effectively.

The artificial bee colony algorithm (ABC), introduced by Karaboga [12] mimics the foraging behavior of nature honey bee. The strength of ABC is the abilities to explore and exploits the problem search region. However, there are still some identified weaknesses regarding these two abilities. Firstly, in ABC, the inactive solution is replaced by randomly generated new solution by scout bees and this to date has induced slow convergence speed (strong exploration) of the search process. Furthermore, the used of weak neighborhood moves in employed and onlooker bees are unable to fine-tune the solution search space (weak exploitation). Hence, a solution is proposed in this study with the aim of attaining the balance between exploration and exploitation abilities of fundamental ABC algorithm.

In this study, global best concepts from particle swarm optimization [5], integration of nelder-mead simplex local search [7] within great deluge algorithm [6] is hybridized with basic ABC algorithm (known as NMGD-ABC). The proposed approach attempts to enhance both exploration and exploitation abilities so that the convergence speed of basic ABC algorithm is improved.

Two timetabling benchmark datasets: Carter un-capacitated examination dataset [1] and Socha course timetabling dataset [2] are used to evaluate the performance of the proposed approach. Experimental results demonstrate the robustness of proposed approach where it capable of solving two different timetabling problems and producing good quality solutions when compared against approaches published in the literature.

2 The hybrid approach

In order to overcome the identified weaknesses in the basic ABC algorithm, a global best concept inspired from particle swarm optimization and integration of nelder-mead simplex

local search within great deluge algorithm is hybridized with basic ABC algorithm (known as NMGD-ABC). Flowcharts for the proposed approach can be seen at Fig. 1. There are four phases in the algorithm: initialization, employed bee, onlooker bee and scout bee phases.

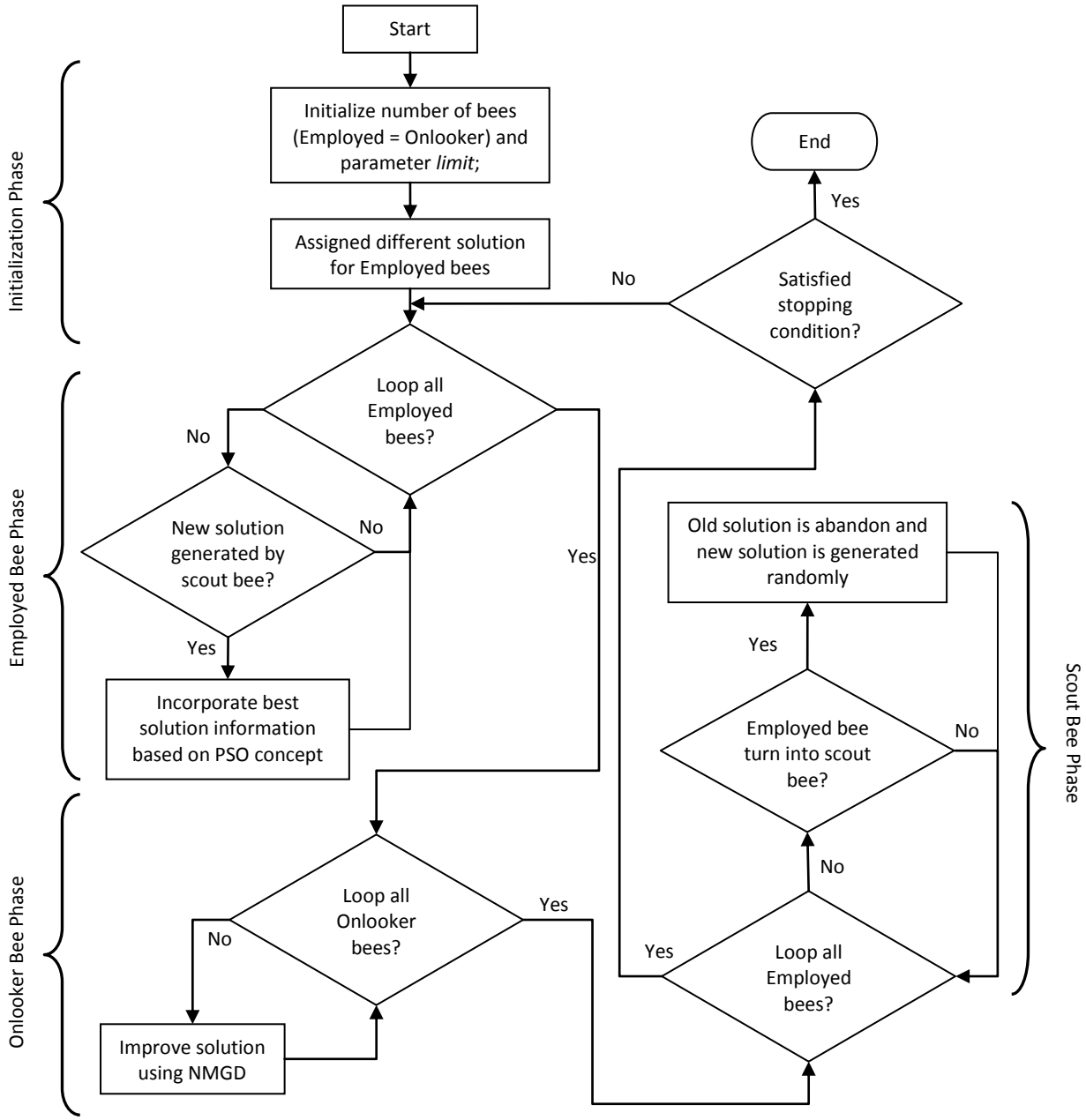


Fig. 1 Flowchart for NMGD-ABC

2.1 Initialization phase

In this phase, the number of bees in the population is initialized. The initial population is constructed using the hybridization of sequential heuristic which are the combination of largest degree, largest enrollment and saturation degree heuristics. Besides these, only the hard constraints are taking into account during the solution construction process.

2.2 Employed bee phase

Global best concept inspired from particle swarm optimization [5] is implemented in the employed bee phase where the information from best solution is incorporated into the new solution generated in the scout bee phase. This can lead the search process to explore promising search region guided by the information from the best solution.

2.2 Onlooker bee phase

In the onlooker bee phase, the great deluge algorithm [6] is implemented with the aim of enhancing the local exploitation ability of the basic ABC algorithm in fine-tuning the solution search region. In addition, nelder-mead simplex local search [7] is also integrated within the great deluge (known as NMGD) so that the estimated quality for great deluge algorithm is intelligently calculated.

2.2 Scout bee phase

In this phase, the inactive solution is identified and discarded. A new solution will be generated randomly and replaces the discarded solution. The inactive solution is determined based on a predefined parameter (*limit*) where if a solution is unable to be improved in a predefined number of iteration (*limit*), it will be discarded.

3 Experimental Results and Conclusion

Experimental results for 30 runs on Carter un-capacitated examination dataset [1] for basic ABC and NMGD-ABC together with the results comparison with the best results in the literature can be seen at Table 1. By referencing to Table 1, it is clear that NMGD-ABC outperforms basic ABC algorithm and producing two new best results (car-f-92 and tre-s-92 instances). In addition, NMGD-ABC obtained a same result with Sabar et al. [8] on uta-s-92 instance.

TABLE 1
Result comparison with best known results

Datasets	ABC		NMGD-ABC		Best known results	
	Best	Avg.	Best	Avg.		
car-f-92	4.80	4.82	**3.89**	4.27	3.90	Sabar et al. [8]
car-s-91	5.60	5.76	4.79	4.85	**4.50**	Burke et al. [9]
ear-f-83	37.47	38.88	33.43	34.48	**29.30**	Caramia et al. [10]
hec-s-92	11.58	11.90	10.49	10.61	**9.20**	Caramia et al. [10]
kfu-s-93	15.25	15.54	13.72	13.76	**13.00**	Burke et al. [9]
lse-f-91	12.48	12.87	10.29	10.39	**9.60**	Caramia et al. [10]
sta-f-83	158.27	159.01	157.07	157.37	**156.90**	Burke et al. [9]
tre-s-92	8.85	8.97	**7.86**	8.04	7.87	Sabar et al. [8]
uta-s-92	3.75	3.89	**3.10**	3.31	**3.10**	Sabar et al. [8]
ute-s-92	27.65	28.41	25.33	26.04	**24.40**	Caramia et al. [10]

yor-f-83	40.21	40.55	36.12	36.67	**34.90**	Burke et al. [9]

TABLE 2
Result comparison with best known results for Socha course timetabling

Datasets	ABC		NMGD-ABC		Best known results	
	Best	Avg.	Best	Avg.		
small 1	25	31.03	**0**	0	**0**	Sabar et al. [8], Turabieh and Abdullah [11]
small 2	32	36.70	**0**	0	**0**	Sabar et al. [8], Turabieh and Abdullah [11]
small 3	23	26.57	**0**	0	**0**	Sabar et al. [8], Turabieh and Abdullah [11]
small 4	15	20.83	**0**	0	**0**	Sabar et al. [8], Turabieh and Abdullah [11]
small 5	24	28.67	**0**	0	**0**	Sabar et al. [8], Turabieh and Abdullah [11]
medium 1	393	433.67	57	70.63	**50**	Turabieh and Abdullah [11]
medium 2	436	470.50	**54**	79.53	70	Turabieh and Abdullah [11]
medium 3	483	499.80	114	132.03	**102**	Turabieh and Abdullah [11]
medium 4	422	443.93	74	82.60	**32**	Turabieh and Abdullah [11]
medium 5	415	455.73	64	75.43	**61**	Turabieh and Abdullah [11]
large	908	991.67	**502**	549.90	523	Sabar et al. [8]

Experimental results on Socha course timetabling dataset [2] can be seen at Table 2. The experiment was run 30 times. Table 2 shows that the NMGD-ABC superiors basic ABC algorithm for all instances. In addition, NMGD-ABC is capable producing zero penalty value for all small instances and producing two new best results on medium 2 and large instances.

In this study, a hybridized approach has been proposed and applied in solving both university exam and course timetabling problems. Through the use of population approaches, the solution exploration process is guided towards promising search region. In addition, the strength of single-solution based approaches has improved the exploitation ability of ABC algorithm in searching local solution space.

Two approaches are hybridized into ABC algorithm: (i) global best concept from particle swarm optimization to guide exploration process in exploring promising search regions, and (ii) integration of nelder-mead simplex search within great deluge algorithm to fine-tune the promising search region.

From the experimental results, it is illustrated that the proposed approach outperforms basic ABC algorithm in searching good quality solutions in the solution search region. In addition, the hybridized approach performs well in solving both university examination and course timetabling problems which demonstrates the generality and robustness of the hybridized approach. It is believed that by combing the strength of exploration from population based approaches and strength of exploitation from single-solution based approaches we are able to improve the solutions searching process of ABC algorithms.

References

1. 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).
2. Socha, K., Joshua, K. 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).
3. 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).
4. 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, J. of Scheduling, 12(1), pp. 55-89 (2009).

5. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory, In: Proceedings of the Sixth International Symposium on Micromachine and Human Science, pp. 39-43 (1995).
6. Dueck G.: New Optimization Heuristics: The great deluge algorithm and the record-to-record travel, Journal of Computational Physics, 104, pp. 86-92 (1993).
7. Nelder, J. A. and Mead, R.: A simplex method for function minimization, The Computer Journal, 7, pp. 308-313 (1965).
8. Sabar, N.R., Ayob, M., Kendall, G. and Qu, R.: A honey-bee mating optimization algorithm for educational timetabling problems, European Journal of Operational Research, 216, 533-543 (2012).
9. Burke, E.K., Eckersley, A.J., McCollum, B., Petrovic, S. and Qu, R.: Hybrid variable neighbourhood approaches to university exam timetabling, European Journal of Operational Research 206, 46-53 (2010).
10. 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).
11. Turabieh, H., Abdullah, S., McCollum, B. and McMullan, P.: Fish swarm intelligent algorithm for the course timetabling problem. In: Yu, J., Greco, S., Lingras, P., Wang, G. and Skowron, A. (eds.) Rough Set and Knowledge Technology. vol. 6401, pp. 588-595. Springer, Heidelberg (2010).
12. Karaboga D.: An idea based on honey bee swarm for numerical optimization. Erciyes University, (2005).

MISTA 2013

A Hybrid of Differential Evolution and Simulated Annealing Algorithms for the Capacitated Arc Routing Problems

Nasser R. Sabar • Masri Ayob • Graham Kendall

Abstract The capacitated arc routing problem (CARP) is a variation of the classic vehicle routing problem in which the objective is to minimize the cost of serving a subset of required arcs without violating vehicle capacity. CARP is an NP-hard combinatorial optimization problem and has many real world applications such as mail delivery, street sweeping and garbage collection. Differential evolution is a population based algorithm that has been shown to be competitive when compared to other methods when tested on continuous optimization problems. In this work, we propose differential evolution as a suitable methodology for combinatorial problems. We use a random key encoding scheme in order to apply differential evolution to combinatorial optimization problems. The exploitation of differential evolution is enhanced by hybridizing it with simulated annealing. Experimental results on CARP benchmark instances demonstrate that the proposed hybrid algorithm obtains very good results, producing new best results for some instances when compared to other state of the art methods.

1. Introduction

The capacitated arc routing problem (CARP), introduced by Golden and Wong [1], is a variation of the classic vehicle routing problem. CARP can be defined as the problem of serving a set of streets using a set of vehicles of a fixed capacity. Given an undirected graph, nodes represent the junctions and arcs represent streets. Arcs are divided into a subset of the required arcs and a subset of the non-required arcs. All required arcs have a non-negative traveling cost and positive demand. Non-required arcs have a non-negative traveling cost only. CARP seeks to find a routing plan to serve all required arcs at minimal traveling cost while satisfying the following constraints [1]:

- Each route begins and ends at the depot.
- Each required arc is served only once.
- The total demand assigned to a vehicle should not exceed the vehicle capacity.

N. R. Sabar • M. Ayob
Data Mining and Optimization Research Group (DMO), Centre for Artificial Intelligent (CAIT), Universiti Kebangsaan Malaysia, 43600 UKM Bangi Selangor, Malaysia.
E-mail: naserdolayme@yahoo.com. masri@ftsm.ukm.my.

G. Kendall
ASAP Research Group, School of Computer Science, The University of Nottingham, Nottingham NG8 1BB, UK.
Email:Graham.Kendall@nottingham.ac.uk.

N. R. Sabar • G. Kendall
The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih Selangor, Malaysia.
E-mail: Graham.Kendall@nottingham.edu.my

CARP has numerous important applications such as mail delivery, street sweeping and garbage collection [1]. CARP is an NP-hard combinatorial optimization problem and thus exact methods are usually only possible for small/medium sized instances. Over the years, many meta-heuristic approaches have been proposed to tackle CARP. Examples of approximate/meta-heuristics that have been utilized for CARP include memetic algorithms [2-3], tabu search [4], variable neighborhood search [5], ant colony optimization [6] and GRASP [7].

Differential evolution (DE) [8] is a population based algorithm which has produced very good results compared to other population based methods when tested on continuous optimization problems. However, DE has largely been utilized on continuous optimization problems, and has focused on exploration only. In this work we propose a random key encoding scheme in order to apply DE to combinatorial optimization problems. Furthermore, motivated by the good performance of local search algorithms on CARP, in particular GRASP and tabu search, we hybridize DE with simulated annealing (SA) [9]. The motivation is that local search methods are good at intensification, whilst population based methods are good at diversification, thus hybridizing them might produce an effective search method. The proposed hybrid algorithm is tested on the CARP benchmark instances and compared with the best methodologies reported in the scientific literature. Our results show that the proposed hybrid algorithm is competitive and is able to produce new best results for some instances.

2. The proposed method

2.1 Differential evolution algorithm

Differential evolution (DE) is a population based method proposed in [8], to solve continuous optimization problems. DE generates a new population of solutions using two operators: mutation and crossover [8]. The mutation operator generates a new solution by combining three different solutions. The crossover operator combines the solution generated by the mutation operator with the initial solution. Then, the generated solution is added to the new population if its quality is better than the initial solution. The basic steps of the DE are as follows [8]:

Step 1: Randomly generate a population of solutions, *NP*.

Step 2: Evaluate the fitness, *f*, of the population.

Step 3: For each solution x_i^G in the current population *NP* (*i* is the solution index and *G* is the current generation) generate a mutated solution (m_i^G) using (1):

$$m_{i,j}^G = x_{1,j}^G + F*(x_{2,j}^G - x_{3,j}^G), \qquad \forall j \in \{1,...,n\} \tag{1}$$

where *j* represent the decision variable, *n* represent the maximum number of decision variables in a given problem instance, *F* is the scaling factor ($F \in [0, 1]$) and x_1^G , x_2^G and x_3^G are three randomly selected solutions from the current population where $x_1^G \neq x_2^G \neq x_3^G$.

Step 4: Recombine the solution generated by a mutation operator (m_i^G) with the target solution (x_i^G) based on the crossover rate *CR* (*CR* ∈ [0, 1]) to generate a new offspring (m_i^{G+1}) as follows (2):

$$m_{i,j}^{G+1} = \begin{cases} m_{i,j}^G \ \ if\ Rand(j) \le CR or\ j = Rnb(i) \\ x_{i,j}^G \ \ if\ Rand(j) > CR and\ j \ne Rnb(i) \end{cases}, \quad \forall j \in \{1,...,n\}, \forall i \in \{1,...,|\ NP\ |\} \qquad (2)$$

where *Rand(j)* is a random number (*Rand(j)* ∈ [0, 1]) selected for the j^{th} decision variable, *Rnb(i)* is a random decision variable index (*Rnb(i)* ∈ {1,…, *n*}). *Rnd* ensure that m_i^{G+1} gets at least one decision variable from m_i^G.

Step 5: Calculate the fitness of m_i^{G+1} and compare it with x_i^G. Replace x_i^G with m_i^{G+1} if m_i^{G+1} fitness is better than x_i^G as follows (3):

$$x_i^{G+1} = \begin{cases} m_i^{G+1} \ \ if\ \ f(m_i^{G+1}) \le f(x_i^G) \\ x_i^G \ \ if\ \ f(m_i^{G+1}) > f(x_i^G) \end{cases}, \quad \forall i \in \{1,...,|\ NP\ |\}$$

(3)

Step 6: If the termination criterion is satisfied (the number of generations), stop and return the best solution. Otherwise, go to **Step 3**.

2.2 The proposed hybrid algorithm for the CARP

2.2.1 DE for CARP

DE was originally proposed to solve continuous optimization problems [8], in which the values of the decision variables can be used directly in the calculation of the fitness function *f*(*x*). In order to apply DE to combinatorial optimization problems, such as CARP, DE solutions need to be converted into discrete values that represent a solution of the CARP. In this work, we use a random key (RK) encoding scheme [10] to represent the solutions of DE. In a RK encoding scheme, solutions are represented by a randomly generated number in the interval [0, 1]. Then, a decoder is applied to convert the RK number of a given solution into a CARP solution. The RK scheme work as follows: first generate a random number, *r*, between zero and one for each decision variable of a given solution. Next, sort the generated random keys in an descending order and map the i^{th} position of the random key with the corresponding decision variable of a given problem instance to generate a solution. For example, assume an instance of CARP that has 6 arcs that need to be served. If we assign numbers for the required arcs from 1 to 6, the decision variables will be (1, 2, 3, 4, 5, 6). Next, generate for each decision variable a random number *r* (*r* ∈ [0, 1])), as shown in Table 1.

Table 1 The decision variables and their corresponding random key

Decision variables	1	2	3	4	5	6
Random key (RK)	0.23	0.45	0.099	0.80	0.65	0.21

To decode Table 1 into a CARP solution, first sort the generated random keys in Table 1 in a descending order, RK: (0.80, 0.65, 0.45, 0.23, 0.21, 0.099). Next, use the indices of the sorted random keys to generate the solution for a given problem instance. Thus, the corresponding sequence for the sorted random keys will be S: (4, 5, 2, 1, 6, 3). Next, we convert the generated sequence into a CARP solution by partitioning a given sequence into a set of planning routes to serve the all required arcs without violating vehicle capacity. For example, if we assume that the demand of all arcs is 1 and the vehicle capacity is 4, then the solution for the above sequence (S: (4, 5, 2, 1, 6, 3)) will be: CARP solution (2 routes): 4 →5→2 and 1→6→3. Therefore, in this work, the proposed DE for the CARP deals with a random key (RK) encoding scheme and the generated random keys by DE are then converted into CARP solutions, which are then evaluated.

2.2.2 The proposed Hybrid algorithm

To further improve the performance of DE, we hybridize it with a simulated annealing algorithm (SA) [9] (denoted as DE-SA). At each generation of the DE, solutions that are generated by the mutation and crossover operators are further improved by SA. Hence, SA takes place after the mutation and crossover operators and before DE population update step.

The proposed hybrid DE-SA works as follows:

1. Convert the solution generated by the DE mutation and crossover operators from the random key encoding scheme to a CARP solution.
2. Call SA to improve the CARP solution.
   SA is an improvement method which starts with an initial solution and iteratively explores its neighborhood solutions using a neighborhood operator. A generated neighborhood solution is accepted if its quality is better than the incumbent solution (in this work, solution quality represents the total traveling distance, where lower the better). However, non-improving neighborhood solutions are accepted based on a probability acceptance function, $R<\exp(-\delta/t)$, where R is a random number between [0, 1], δ is the change in the solution quality and t represents the temperature. The ratio of accepted moves to worse solutions is controlled by a temperature t which gradually decreases by β during the search process. In this work, a neighborhood solution is generated by applying one of the following move operators which is selected at random: 2-opt, 2-opt*, random move and random swap.
   a. If termination criterion of SA is satisfied, then decode the CARP solution into random key encoding scheme, calculate the quality of the improved solution, and update the DE population of solutions.
3. Check DE stopping criterion (number of generations). If it satisfied, stop and returns the best solution. Otherwise, start a new generation.

3. Computational results

The performance of the proposed DE-SA is tested on the *egl* CARP benchmark which comprises 24 instances of different required arcs and capacity [11]. Table 2 lists the parameter settings of the DE-SA, which were set by an initial set of experiments.

Table 2 DE-SA parameter settings

#	Parameter	Value
1-	No. Of generations	500
2-	Population size, *NP*	30
3-	Scaling factor, *F*	0.1
4-	Crossover rate, *CR*	0.9
5-	SA initial temperature, *t*	*t*=50% of the initial solution cost
6-	SA cooling schedule, β	β=0.85
7-	SA final temperature, t_0	t_0=0

Table 3 presents the best results obtained by DE-SA over 31 independent runs along with the best results obtained by the best known methods in the literature. Best results are highlighted with a grey background, with new best results from our proposed algorithm being also shown in bold. In this table, |V|, $|E_R|$, |E|, LB and BKV indicate the number of vertices, the number of required arcs, the total number of edges, the lower bound and the best known results, respectively. Please note that values in Table 3 represent the total traveling distances (the lower the better).

As can be seen from Table 3, out of 24 instances, DE-SA matches the best known results in the literature on 14 instances and produced a better quality for 4 instances (E4-B, S2-A, S2-C and S3-B) compared to the state of the art methods. With respect to the computation time, Table 4 reveals that DE-SA average time is lower than the compared methods.

Table 3 The best results of DE-SA compared to state of the art methods

Instances	|V|	|ER|	|E|	LB	BKV	DE-SA	PMA	IMA	TS	VNS	ACO	GRASP
E1-A	77	51	98	3548	3548	3548	3548	3548	3548	3548	3548	3548
E1-B	77	51	98	4498	4498	4498	4498	4498	4533	4498	4498	4498
E1-C	77	51	98	5566	5595	5595	5595	5595	5595	5595	5595	5595
E2-A	77	72	98	5018	5018	5018	5018	5018	5018	5018	5018	5018
E2-B	77	72	98	6305	6317	6317	6317	6317	6343	6321	6317	6317
E2-C	77	72	98	8243	8335	8335	8335	8335	8347	8335	8335	8335
E3-A	77	87	98	5898	5898	5898	5898	5898	5902	5898	5898	5898
E3-B	77	87	98	7704	7777	7775	7775	7777	7816	7775	7777	7777
E3-C	77	87	98	10163	10292	10292	10292	10292	10309	10292	10292	10292
E4-A	77	98	98	6408	6456	6456	6444	6456	6473	6446	6465	6444
E4-B	77	98	98	8884	8991	**8986**	8988	8991	9063	9004	8990	9002
E4-C	77	98	98	11427	11587	11596	11594	11587	11627	11652	11624	11626
S1-A	140	75	190	5018	5018	5018	5018	5018	5072	5018	5018	5018
S1-B	140	75	190	6384	6388	6388	6388	6388	6388	6388	6388	6388
S1-C	140	75	190	8493	8518	8518	8518	8518	8535	8518	8518	8518
S2-A	140	147	190	9824	9911	**9901**	9908	9911	10038	9944	9895	9903
S2-B	140	147	190	12968	13150	13150	13153	13150	13178	13167	13194	13169
S2-C	140	147	190	16353	16489	**16440**	16442	16489	16505	16491	16461	16442
S3-A	140	159	190	10143	10260	10260	10229	10261	10451	10259	10249	10221
S3-B	140	159	190	13616	13807	**13690**	13711	13813	13981	13751	13786	13694
S3-C	140	159	190	17100	17234	17234	17244	17288	17346	17299	17269	17221
S4-A	140	190	190	12143	12341	12341	12323	12348	12462	12375	12324	12297
S4-B	140	190	190	16093	16345	16345	16331	16345	16490	16353	16428	16333
S4-C	140	190	190	20375	20556	20556	20578	20556	20733	20640	20595	20563

Note: PMA- probabilistic memetic algorithm [3]. **IMA**- improved memetic algorithm [2]. **TS**- tabu search [4]. **VNS**- variable neighborhood search [5]. **ACO**- ant colony optimization [6]. **GRASP**- greedy randomize adaptive search procedure [7].

Table 4 The average execution time of DE-SA and other methods

Algorithm	*DE-SA*	*PMA*	*IMA*	*TS*	*VNS*	*ACO*	*GRASP*
Average time (s)	185.14	-	307.7	291.4	503.2	503.0	798.58

The obtained results demonstrate that DE-SA is very competitive when compared to other state of the art methodologies. Moreover, DE-SA finds new best results for 4 out of 24 CARP instances. These positive results support the fact that hybridizing a population based method

(DE in this work) and a local search algorithm (SA in this work) produces an effective search methodology.

4. Conclusion

In this work, we have proposed a hybrid algorithm that combines differential evolution with simulated annealing for the capacitated arc routing problems. Differential evolution is a population based method that focuses on exploration of the search space. Simulated annealing algorithm is a local search algorithm that focuses on exploitation of the search space. The proposed algorithm is applied to the capacitated arc routing problem benchmark instances. Results demonstrate that the proposed hybrid algorithm is very competitive and manages to produce new best results for four instances when compared to other methodologies in the scientific literature.

References

1. Golden, B. L. and Wong, R. T., "Capacitated arc routing problems", Networks, 11, pp. 305-315, 1981.
2. Y. Mei, K. Tang, and X. Yao, “Improved memetic algorithm for capacitated arc routing problem,” IEEE Congress on Evolutionary Computation, pp. 1699–1706, 2009.
3. L. Feng, Y. S. Ong, Q. H. Nguyen, and A. H. Tan, “Towards probabilistic memetic algorithm: An initial study on capacitated arc routing problem,” IEEE Congress on Evolutionary Computation 2010, pp. 18–23, 2010.
4. Brandao J, Eglese R. A deterministic tabu search algorithm for the capacitated arc routing problem. Computers and Operations Research 2008; 35:1112–26.
5. Polacek M, Doerner K, Hartl R, Maniezzo V. A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. Journal of Heuristics 2008; 14:405–23.
6. Santos L, Coutinho-Rodrigues J, Current JR. An improved ant colony optimization based algorithm for the capacitated arc routing problem. Transportation Research Part B: Methodological 2010; 44(2):246–66.
7. Usberti, F. L., França, P. M. and França, A. L. M.: Grasp with evolutionary path-relinking for the capacitated arc routing problem, Computers and Operations Research 2011, in press. doi: 10.1016/j.cor.2011.10.014.
8. Storn, R.; Price, K. (1997). "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces". Journal of Global Optimization 11: 341–359.
9. Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. (1983). "Optimization by Simulated Annealing". Science 220 (4598): 671–680.
10. J. C. Bean. Genetics and random keys for sequencing and optimization. Technical Report 92-43, Dept. of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI 48109, 1992.
11. L.Y.O. Li and R.W. Eglese. An Interactive Algorithm for Vehicle Routing for Winter-Gritting. Journal of the Operational Research Society, 1996 47, pp. 217-228.

MISTA 2013

Flexibility Metrics for Time and Resource Constrained Scheduling Problems

Tomas Klos · Michel Wilson · Cees Witteveen

Abstract Flexibility is an important and much wanted property of solutions to scheduling problems. Intuitively, the more flexible a solution is, the more freedom we have in choosing a starting time of an activity without having to change the existing schedule in an essential way. In this paper we discuss flexible solutions to the Resource Constrained Project Scheduling Problem (RCPSP). In the past years much attention has been given to a special method to obtain flexible solutions to the RCPSP, which employs Partial Order Schedules (POS). Basically, this method reduces an RCPSP instance to a temporal constraint specification, in this case an instance of the Simple Temporal Problem (STP). Instead of encoding a single solution, this STP representation encodes a number of solutions to the original RCPSP-instance. Not surprisingly, much research has been devoted to providing flexibility metrics for the STP. After reviewing some existing flexibility metrics, however, we conclude that these fail to capture the correlation between events specified in the STP, resulting in an overestimation of the available flexibility in the system. Therefore, we propose an intuitively more acceptable new flexibility metric for general linear constraint systems. We also show that this flexibility metric can be used in autonomous scheduling: Whenever a scheduling problem has to be solved by more than one independent actor, we would like to let these actors solve their part of the problem independently without violating the original constraints. This is the so-called *decoupling problem*. As a byproduct of the flexibility computation, we get a decoupling of the instance almost for free: for every possible partitioning of the event space among k agents, a decoupling can be computed in linear-time. Even more importantly, we show that—contrary to popular belief—such a decomposition does not need to affect the flexibility of the original linear constraint system.

Tomas Klos
Delft University
E-mail: T.B.Klos@tudelft.nl

Michel Wilson
Delft University
E-mail: M.Wilson@tudelft.nl

Cees Witteveen
Delft University
E-mail: C.Witteveen@tudelft.nl

1 Introduction

Scheduling problems occur in many application domains such as transportation, process industry, health and education. Informally, a scheduling problem arises if (i) there is a set of events whose (starting) times are subject to temporal as well as resource constraints and (ii) we have to determine the starting times of these events such that all constraints are satisfied. Usually, we are not satisfied with determining a single schedule, i.e., an assignment σ determining for every event x a fixed starting time $\sigma(x)$. First, because of outside disturbances (e.g., unforeseen delays, resources that are not available), it may not be possible to start x at the exact time at which it has to be scheduled. Then it is essential for the scheduling agent to be able to adapt the current schedule. Since during execution there is not much time for rescheduling, alternative starting times should be computable in a very short time or, even better, known in advance. Therefore, instead of offering a single fixed schedule as a solution, one would like to find a representation encoding a number of (easy to retrieve) solutions. Intuitively, the more alternative schedules such an encoding would provide, the more flexibility we have.

A second reason to ask for flexibility occurs when the problem is distributed among *multiple actors* who want to independently solve their part of the overall scheduling problem. For example, in a resource constrained project scheduling problem (RCPSP), activities might be distributed over several companies. Typically, these companies want to solve their part of the scheduling problem independently of the others, based on e.g. their own preferences. To guarantee independent scheduling, we face a *decomposition* problem: how can we ensure that all individual (i.e. company) solutions can be merged into a common solution that satisfies all the original constraints? One method that has been applied to temporal scheduling problems (scheduling problems without resource constraints) is known as *temporal decoupling* [1,2]. Here the original scheduling problem is decomposed into a number of subproblems, such that any schedule for a subproblem can be merged with any schedule for each of the other subproblems, to constitute a feasible schedule of the total problem.

As has been pointed out by various authors [1,3–5], temporal decoupling may affect the total flexibility of the system (as measured by existing flexibility metrics). The question then arises how we can minimize this flexibility loss due to temporal decoupling. Hence, we would also like to (i) optimize the decoupling with respect to flexibility and (ii) investigate the loss of flexibility due to decoupling, if any.

With flexibility as our main topic, in this paper we address three specific topics.

1. First of all, we propose a new flexibility metric for the RCPSP. This flexibility metric will be based on the encoding of solutions of an RCPSP in terms of a temporal specification called a Simple Temporal Problem (STP) [6]. This implies that we also compare our flexibility approach with other flexibility measures for STP's.
2. Secondly, to show the generality of this approach to flexibility, we derive a flexibility metric for general linear constraint systems far extending the temporal specification for STP's.
3. Finally, we investigate the consequences of using our flexibility metric for decoupling in linear constraint systems and especially the STP. Basically, we will follow the same approach as in answering the first question: The decoupling will be applied to the reduction of an RCPSP to a temporal specification. We then investigate the

flexibility of the decoupled system and compare the resulting flexibility with the flexibility of the original system.

Our approach to addressing these issues is the following. First, we present a brief overview of existing research on solving the RCPSP by reducing an RCPSP instance to an instance of the Simple Temporal Problem. An STP derived from an RCPSP is a temporal constraint system ensuring that any solution to it will satisfy not only all temporal constraints but also all resource constraints occurring in the RCPSP: The latter are satisfied in the construction of the STP. The advantage of this reduction is that an STP encodes a set of solutions and every solution can be retrieved in polynomial time. Our problem then comes down to specifying a suitable flexibility metric for STP's. After a brief discussion of existing metrics, we present our new metric.

In order to show the generality we show how our flexibility metric applies to any system of linear equations of the form $\mathbf{Ax} \leq \mathbf{b}$ and we discuss the decoupling problem for general linear equations. Surprisingly, the method we use to specify the flexibility of linear equations allows us to compute a decoupling for an STP that does not induce any loss of flexibility due to decoupling and can be obtained in linear time (if the flexibility of the system has already been computed). This result shows that temporal decoupling can be achieved both efficiently and without loss of flexibility.

2 The RCPSP and its reduction to temporal constraints

The scheduling problem we will discuss is the RCPSP (see [7] for an overview of the RCPSP and many variants). The basic ingredients of an RCPSP instance are a set $X = \{x_1, x_2, \ldots, x_n\}$ of activities and a set $R = \{r_1, r_2, \ldots, r_m\}$ of renewable resources. Execution of activities is subject to both time and resource constraints. More specifically,

1. for every $x_i \in X$ there is a duration (processing time) p_i;
2. for every activity x_i there are unary temporal constraints specifying release and due date:
$$x_i \geq s_i,$$
$$x_i \leq d_i;$$
3. for every pair of activities x_i and x_j there are binary temporal constraints specifying minimal and maximal time lags between the start times of x_i and x_j:
$$l_{ij}^{min} \leq x_j - x_i \leq l_{ij}^{max};$$
4. for every resource $r_j \in R$ its capacity $R_j \geq 1$ is specified and for every activity x_i the amount $R_{ij} \geq 0$ of resource r_j that is needed to execute x_i is given. At any time t the total amount $\sum_i R_{i,k}$ of a resource r_k, needed by all activities i in progress at time t should not exceed the resource's capacity R_k.

The objective for such an RCPSP is to determine a schedule $\sigma : A \to \mathbb{R}^{\geq 0}$ for the activities in A such that all time and resource constraints are satisfied.

Note that the first three requirements pertain to a set C_T of (linear) time constraints for activities and the last requirement specifies a set C_R of resource constraints that have to be met. An RCPSP instance S then can be considered as a tuple $S = (A, R, C_T, C_R)$.

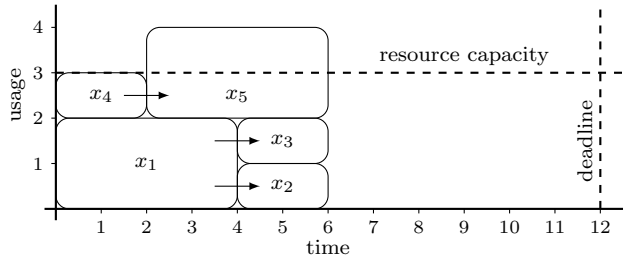


Fig. 1 Resource view of a simple RCPSP instance with five activities. Temporal constraints are depicted using arrows with single lines. The release and due times are not shown in this picture.

Example 1 In Figure 1 an instance of the RCPSP is given, with five activities and one resource. The processing time of the tasks is indicated on the horizontal axis, and the resource usage on the vertical axis. The problem includes temporal constraints of the form $p_i \leq x_j - x_i \leq \infty$ (i.e., constraints of the form "start x_j after x_i is completed"), these are depicted by arrows, and a deadline for all tasks at $t = 12$, indicated by the dashed vertical line. The dashed horizontal line depicts the capacity of the only resource given; it is clear that its capacity is exceeded by the simultaneous requirements of the sets of activities $\{x_1, x_5\}$ and $\{x_2, x_3, x_5\}$.

There exists numerous solvers for the RCPSP ranging from optimal exact methods [8,9] to heuristics [10–12] and meta-heuristic approaches [13]. It is not our aim to discuss these solvers, since we are not interested in the solutions for RCPSP per se, but in finding a suitable metric to determine the flexibility of an RCPSP instance, that is an evaluation of the amount of freedom we have in choosing starting times of the activities without violating any of the constraints.

So instead of looking for a single solution of an RCPSP, we investigate representation mechanisms for RCPSP that allow us to describe this flexibility in an efficient way. An interesting approach in this regard was proposed by Policella et al. [14,11]. Their approach comes down to a reduction of an RCPSP instance $S = (A, R, C_T, C_R)$ to a temporal constraint system $S' = (A, C_T \cup C_T^{'})$ where all resource constraints have been translated into a set of additional time constraints $C_T^{'}$ in such a way that every solution for S' is a solution to the original RCPSP instance S. The essential idea here is that whenever there is a resource conflict between activities x_i and x_j w.r.t. a resource r_k, such a conflict can be resolved by letting x_i precede x_j (by 'posting' the temporal constraint $x_i + d_i \leq x_j$, sometimes written as the precedence constraint $x_i \prec x_j$) or vice versa. By means of this so-called *peak-shaving* procedure, we can eliminate resource conflicts by adding precedence constraints.[1] The advantage of this so-called *constraint posting method* is that by means of the resulting temporal constraint specification S', called a *Partial Order Schedule* (POS) by the authors, not a single solution, but a set of solutions is encoded in the temporal specification. Since quite some research has been done into determining the flexibility of these temporal specifications, it is our goal to evaluate the flexibility of an RCPSP instance S by using a suitable flexibility metric for the resulting temporal specification S'. Before we discuss these flexibility metrics, we first deal with some details of the resulting temporal specification.

[1] In this paper, we do not discuss the details and variants of the constraint posting method.

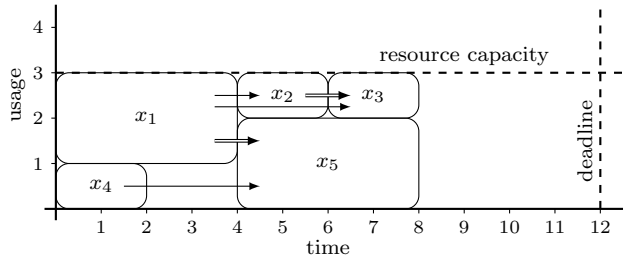


Fig. 2 The RCPSP instance from Figure 1, with two added constraints (depicted using double lines), rendering the problem resource-feasible.

Example 2 In Figure 2, Example 1 is reduced to a purely temporal specification, by adding the two precedence constraints $x_2 \prec x_3$ and $x_1 \prec x_5$. Due to these precedence constraints, the resource violations have been resolved. Note that due to the structure of this reduction, *any* valid temporal assignment to the activities is also a resource-feasible schedule of the original instance.

2.1 Simple Temporal Plans

By applying the constraint posting technique, an RCPSP instance is reduced to an instance of the Simple Temporal Problem (STP) [6,15]. In general, an instance of the STP is a pair $S = (X, C)$, where $X = \{x_1, \ldots, x_n\}$ is a set of time-point variables (events) and C is a finite set of binary difference constraints on X, each constraint having the form $x_j - x_i \leq c_{ji}$, for some real number c_{ji}. Here, the variable x_0, also called z (for the 'zero' time-point), is added to the set of the n variables of interest to be able to express absolute (unary) time constraints $x_i \leq c$, such as the constraints for release times and due dates, as binary constraints $x_i - x_0 \leq c$. Given S, we want to find a schedule for the events in X, that is a function $\sigma : X \to \mathbb{R}^{\geq 0}$, assigning a non-negative real value (time-points) to each event in X, such that all constraints in C are satisfied. If such a schedule exists, we say that the STP is consistent. Without loss of generality, in the remaining of the paper, we simply assume that an STP is consistent.[2]

Often we represent an STP instance $S = (X, C)$ as a directed labeled graph G_S where the vertices represent events in X: for every node x_i, $i > 0$, there is a label $[a, b]$ representing the time constraints $x_0 - x_i \leq -a$ and $x_i - x_0 \leq b$. For every pair of constraint in C pertaining to the pair of variables (x_i, x_j), $i, j > 0$, the graph contains a directed edge $e = (x, y)$, with a label $[c, d]$ representing the constraints $x - y \leq -c$ and $y - x \leq d$. The graph G_S associated with an instance of the STP is called a Simple Temporal Network (STN). In this paper, we will use both representations interchangeably.

The following proposition is a summary of two relevant properties that can be proven to hold for STPs and their schedules [15]:

Proposition 1 *Let $S = (X, C)$ be an STP. Then for every $x \in X$ the earliest starting time $est(x)$ and the latest starting time $lst(x)$ can be determined in $O(n^3)$*

[2] Consistency of an STP can be determined in low-order polynomial time [6].

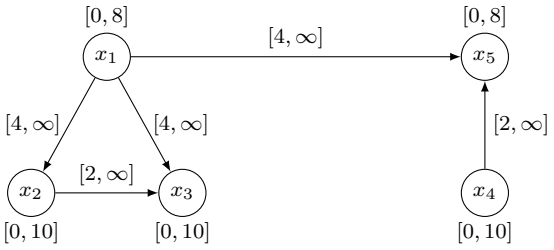


Fig. 3 STN corresponding to the RCPSP shown in Figure 2.

time where $n = |X|$*. For every schedule* σ *for* S*, and every* $x \in X$*, it holds that* $\sigma(x) \in [est(x), lst(x)]$ *and, given any* $x \in X$ *and* $v \in [est(x), lst(x)]$*, there exists a schedule* σ *for* S *such that* $\sigma(x) = v$.[3]

Since we want to solve scheduling problems using STPs, we assume that for each event x there is a *finite horizon* for scheduling it. That means that for each STP $S = (X, C)$ we assume that there exists a finite constant h_S such that for all $x \in X$ it holds that $x \leq h_S$ can be derived from C. This avoids the use of unbounded time intervals (e.g. $x_i - x_j \leq \infty$).

3 Flexibility metrics for STNs

Flexibility refers to the ability to adapt. So in the context of providing a solution to an STN, we may interpret our flexibility in scheduling a task or event as the amount of freedom we have in assigning a starting time to the task. In order to capture this intuitive idea of flexibility a number of flexibility measures for STNs have been proposed in the literature e.g. [17,10,18,1,14], all of which are based on easy to compute characteristics of an STN such as the 'slack' in an STN.[4] Basically, these approaches based on slack in an STN define the flexibility of an STN S as the sum of slack associated with each time point $x \in X$. Here, the slack associated with an event x equals the difference between the latest time x might be scheduled and the earliest time x is allowed to be scheduled:

$$\mathit{flex}_s(S) = \sum_{x \in X} \mathit{slack}(x) = \sum_{x \in X} (lst(x) - est(x))$$

This makes sense, because we know that by Proposition 1, for every individual value $v \in [est(x), lst(x)]$, there exists a schedule σ for the STN S such that $\sigma(x) = v$. Therefore, we are certain that this assignment can be extended to a complete schedule σ for S. Moreover, by Proposition 1 again, we easily derive that $\mathit{flex}_s(S)$ can be computed in an efficient way ($O(n^3)$ time). However, although for every *individual* event x, $\mathit{slack}(x)$ offers a precise measure of its flexibility, using the sum $\mathit{flex}_s(S)$ as the flexibility inherent in the instance S has a serious disadvantage, due to *correlations* that might exist between starting times of *multiple* events as a result of the constraints

[3] In fact it can be shown that there exists a simple backtrack-free polynomial-time algorithm to construct an arbitrary schedule σ for S.

[4] Not all these measures are completely identical, but all have been derived from a slack-based metric, basically suffering from the same disadvantage as we will see.

in C. As we will see these correlations might induce a serious overestimation of the 'real' flexibility of an STN. To see this, consider the following example.

Example 3 Consider the STNs S_1 and S_2 (see Figure 4) where S_1 specifies the *concurrent* execution of $k = 3$ events within 5 time units, while S_2 specifies the *sequential* execution of $k = 3$ consecutive events, i.e., x_3 should occur before x_2, and x_2 should occur before x_1, and all should occur within 5 time units.

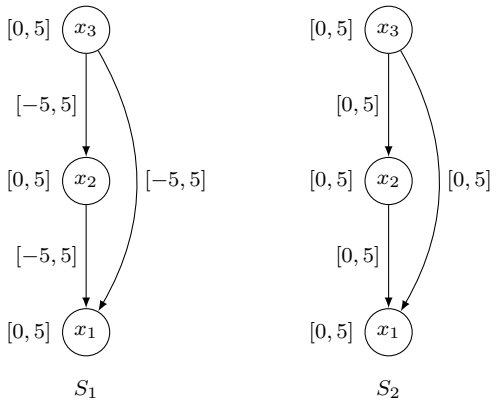


Fig. 4 Two STNs each specifying three events x_1, x_2 and x_3 in the interval $[0, 5]$. Here, S_1 (on the left) specifies the *concurrent* execution of these events in the interval $[0, 5]$, while S_2 encodes their *sequential* execution $x_3 \leq x_2 \leq x_1$.

Note that in both S_1 and S_2, for each $i \in \{1, 2, 3\}$, $est(t_i) = 0$ and $lst(t_i) = 5$. Hence,

$$\mathit{flex}_s(S_1) = \mathit{flex}_s(S_2) = 3 \times 5 = 15.$$

So according to flex_s, S_1 and S_2 have the same flexibility. This is counterintuitive: We can select a starting time for each event x_i in S_1 between 0 and 5 independently from the other events. So the total flexibility in S_1, expressing the amount of freedom we have in selecting starting times for all events equals $5 \times 3 = 15$. On the other hand, fixing the starting time of an event x_i in S_2 $(i = 3, 2)$, influences the freedom we have for assigning starting times of the next events x_j $(j < i)$. More precisely, if (v_1, v_2, v_3) are the latest starting times for the 3 events in S_2, then every solution satisfies $0 \leq v_3 \leq v_2 \leq v_1 = 5$. Hence, we would like a flexibility metric that measures x_3's flexibility as $v_3 - 0$, x_2's flexibility as $v_2 - v_3$ and x_1's flexibility as $5 - v_2$. So, the total flexibility of S_2 should come out as

$$\mathit{flex}(S_2) = v_3 + (v_2 - v_3) + (5 - v_2) = 5.$$

Therefore, by our interpretation of flexibility, the flexibility of S_2 should be only $\frac{1}{3}$ of the flexibility of S_1.

Let us generalize S_1 and S_2 to systems containing k concurrent and sequential events, respectively. Then we would like to have a flexibility metric that measures the freedom to schedule in S_1 as $\mathit{flex}(S_1) = 5 \times k$, while the flexibility of S_2 should be

independent of k and remain $\mathit{flex}(S_2) = 5$. Hence, if k goes to infinity, we want the ratio of the flexibilities of S_1 and S_2 to go to infinity as well:

$$\lim_{k\to\infty} \frac{\mathit{flex}(S_1)}{\mathit{flex}(S_2)} = \lim_{k\to\infty} \frac{5 \times k}{5} = \infty,$$

However, observe that in this case their ratio according to flex_s is exactly 1:

$$\lim_{k\to\infty} \frac{\mathit{flex}_s(S_1)}{\mathit{flex}_s(S_2)} = \frac{5 \times k}{5 \times k} = 1.$$

Therefore, we conclude that flex_s overestimate the flexibility of STNs similar to S_2. This also holds for similar flexibility metrics based on slack of individual events.

4 Towards a Suitable Flexibility Metric for Linear Constraints

Analyzing the failure of $\mathit{flex}_s(S)$ to accurately measure the flexibility of an STN S reveals as its main shortcoming that it does not capture the correlations between the allowed schedule-time intervals for the events in X, if such correlations exist. In this section we will develop a new flexibility metric that does take into account these correlations. In developing our desired flexibility metric we will be as general as possible. So we will not limit the flexibility metric to STN's, but instead consider general systems of linear constraints. Such a system consists of a set of variables X and a set of constraints c_i each of the form

$$a_{i,1}x_1 + a_{i,2}x_2 + \ldots + a_{i,n}x_n \leq b_i$$

Such systems can be more compactly represented by matrix inequalities. Therefore, we represent a set S of linear constraints as the following matrix inequality

$$\mathbf{Ax} \leq \mathbf{b}, \tag{1}$$

where $\mathbf{A}$ is an $m \times n$ matrix of coefficients, $\mathbf{x}$ is an $n \times 1$ vector of variables and $\mathbf{b}$ is an $n \times 1$ vector of coefficients.[5] A solution σ to such a constraint system S is an assignment $\sigma(\mathbf{x}) = \mathbf{v}$ where $\mathbf{v}$ is a constant vector such that $\mathbf{Av} \leq \mathbf{b}$. Intuitively, the flexibility of such a set S is closely related to the set of allowable solutions for S. Therefore, we would like to derive a compact representation for a *set of solutions* Σ of S. The size of this solution space would then express the flexibility of S. A solution space can be generated by starting with the flexibility associated with a single variable x_i in $\mathbf{x}$. Instead of specifying a single value, we specify an interval $[v_i^-, v_i^+]$ of values $v_i^- \leq v_i^+$ for x_i, such that for every solution σ for S, whenever we assign to x_i any value $v \in [v_i^-, v_i^+]$, the resulting assignment is still a solution.

Definition 1 Let S be a set of linear equations $\mathbf{Ax} \leq \mathbf{b}$ and $[v_i^-, v_i^+]$ be an interval for variable $x_i \in \mathbf{x}$. Then $[v_i^-, v_i^+]$ is a *solution interval* for x_i iff for any solution $\sigma(\mathbf{x}) = \mathbf{v}$ for S, the set $\{\sigma[x_i = v] : v \in [v_i^-, v_i^+]\}$ is a set Σ of solutions of S.

[5] Note that an STN is a special case of such a system, containing only linear (difference) constraints of the form $x_i - x_j \leq b$. While we need the variable z in an STN to encode unary constraints, we can encode these directly in systems of linear constraints, so we need just n variables in this context.

Here, $\sigma[x_i = v]$ denotes the assignment σ' that differs from σ only in assigning the value v to variable x_i.

If we have solution intervals $[v_i^-, v_i^+]$ for all the variables x_i in $\mathbf{x}$, we compactly represent the end points of the intervals by the two vectors $\mathbf{v}^- = (v_i^-)$ and $\mathbf{v}^+ = (v_i^+)$. We observe that the following proposition holds by the definition of a solution interval:

Proposition 2 *Suppose that for every $i = 1, 2 \ldots, n$, $[v_i^-, v_i^+]$ is a solution interval for x_i. Then the set*

$$\Sigma = \{\sigma : \mathbf{v}^- \leq \sigma(\mathbf{x}) \leq \mathbf{v}^+\}$$

is a set of solutions of S.

In general, such a set of solutions Σ obtained from solution intervals is an *interval solution* for S, and is defined as follows.

Definition 2 Let $\mathbf{Ax} \leq \mathbf{b}$ a set of linear constraints. Then $I_{\mathbf{x}} = [\mathbf{v}^-, \mathbf{v}^+]$ is an interval solution for $\mathbf{Ax} \leq \mathbf{b}$ if for every $\mathbf{v}$ such that $\mathbf{v}^- \leq \mathbf{v} \leq \mathbf{v}^+$ it holds that $\mathbf{Av} \leq \mathbf{b}$.

Example 4 Consider the STN S_2 presented in Example 4. This STN can be represented by the following system $\mathbf{Ax} \leq \mathbf{b}$ of constraints[6].

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 5 \\ 5 \\ 5 \\ 0 \\ 0 \\ 0 \\ 5 \\ 5 \\ 5 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{2}$$

One possible solution to this system is the vector

$$\mathbf{x} = \begin{bmatrix} 4 \\ 3 \\ 1 \end{bmatrix}, \tag{3}$$

which we can interpret as an assignment of times to the variables that satisfies the inequalities $x_3 \leq x_2 \leq x_1$ encoded in S_2.

A possible interval solution $I_x = [\mathbf{v}^-, \mathbf{v}^+]$ for S is

$$\mathbf{v}^- = \begin{bmatrix} 3 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{v}^+ = \begin{bmatrix} 4 \\ 3 \\ 1 \end{bmatrix}. \tag{4}$$

[6] Note that the time reference variable x_0, having the value 0 has been omitted, since unary constraints can be specified directly.

Since an interval solution specifies a set of solutions, we would like to base our flexibility metric upon such an interval solution. Let us consider the size

$$|I_{\mathbf{x}}| = \sum_{i=1}^{n} (v_i^+ - v_i^-)$$

of an interval solution as a measure of the flexibility associated with $I_{\mathbf{x}}$. In order to obtain a flexibility metric for the underlying system of constraints, it seems natural to consider interval solutions with *maximum* flexibility. So we define *the flexibility of the system* S of linear constraints $\mathbf{Ax} \leq \mathbf{b}$ as the maximum value of $|I_{\mathbf{x}}|$ over all interval solutions $I_{\mathbf{x}}$ of $\mathbf{Ax} \leq \mathbf{b}$:

Definition 3 Let S be a set of linear constraints of the form $\mathbf{Ax} \leq \mathbf{b}$. We define the flexibility of S, denoted by $\mathit{flex}(S)$ as

$$\mathit{flex}(S) = \max\{|I_{\mathbf{x}}| : I_{\mathbf{x}} \text{ is an interval solution for } \mathbf{Ax} \leq \mathbf{b}\}$$

Our goal is to have flexibility metric that is easy to compute. Therefore, we will show that $flex(S)$ can be computed in polynomial time.

First of all, note that if $I_{\mathbf{x}} = [\mathbf{v}^-, \mathbf{v}^+]$ is an interval solution, we have

$$\mathbf{Av}^+ \leq \mathbf{b} \quad \text{and} \quad \mathbf{Av}^- \leq \mathbf{b}$$

Now, given the matrix $\mathbf{A} = [a_{i,j}]$ let us define two matrices $\mathbf{A}^+ = [a_{i,j}^+]$ and $\mathbf{A}^- = [a_{i,j}^-]$ as follows:

$$a_{i,j}^+ = \begin{cases} a_{i,j} & \text{if } a_{i,j} > 0, \\ 0 & \text{else.} \end{cases} \tag{5}$$

and

$$a_{i,j}^- = \begin{cases} a_{i,j} & \text{if } a_{i,j} < 0, \\ 0 & \text{else.} \end{cases} \tag{6}$$

The main step in showing that the flexibility can be computed efficiently is provided by the following proposition, which characterises an interval solution in terms of a single solution to a special set of linear equations:

Proposition 3 *$I_{\mathbf{x}} = [\mathbf{v}^-, \mathbf{v}^+]$ is an interval solution for $\mathbf{Ax} \leq \mathbf{b}$ iff $\mathbf{A}^+\mathbf{v}^+ + \mathbf{A}^-\mathbf{v}^- \leq \mathbf{b}$.*

Proof Suppose $I_{\mathbf{x}}$ is an interval solution and, on the contrary, assume that $\mathbf{A}^+\mathbf{v}^+ + \mathbf{A}^-\mathbf{v}^- \leq \mathbf{b}$ does not hold. Then there exists at least one $1 \leq i \leq n$ such that[7] $(\mathbf{A}^+)_i\mathbf{v}^+ + (\mathbf{A}^-)_i\mathbf{v}^- > (\mathbf{b})_i$. Consider the following vector $\mathbf{v} = (v_j)_{j=1}^n$ where

$$v_j = \begin{cases} v_j^+ & \text{if } a_{i,j}^+ > 0 \\ v_j^- & \text{if } a_{i,j}^+ \leq 0 \end{cases} \tag{7}$$

Then, by definition of $\mathbf{A}^+$ and $\mathbf{A}^-$ we have

$$(\mathbf{A}^+)_i\mathbf{v} + (\mathbf{A}^-)_i\mathbf{v} > (\mathbf{b})_i$$

[7] We indicate the i-th row of a matrix $\mathbf{A}$ by $(\mathbf{A})_i$. Likewise, the i-th entry of a vector $\mathbf{b}$ is indicated by $(\mathbf{b})_i$.

However, since $\mathbf{v}^- \leq \mathbf{v} \leq \mathbf{v}^+$ and $I_{\mathbf{x}}$ is an interval solution, we have

$$(\mathbf{A}^+)_i\mathbf{v} + (\mathbf{A}^-)_i\mathbf{v} \leq (\mathbf{b})_i$$

and arrive at a contradiction. Therefore, $\mathbf{A}^+\mathbf{v}^+ + \mathbf{A}^-\mathbf{v}^- \leq \mathbf{b}$ holds.

Conversely, suppose that $\mathbf{A}^+\mathbf{v}^+ + \mathbf{A}^-\mathbf{v}^- \leq \mathbf{b}$. Take any $\mathbf{v}$ such that $\mathbf{v}^- \leq \mathbf{v} \leq \mathbf{v}^+$. Since $\mathbf{v} \geq \mathbf{v}^-$, it follows that $\mathbf{A}^-\mathbf{v} \leq \mathbf{A}^-\mathbf{v}^-$ and, since $\mathbf{v} \leq \mathbf{v}^+$, we have $\mathbf{A}^+\mathbf{v} \leq \mathbf{A}\mathbf{v}^+$. Hence,

$$\mathbf{A}\mathbf{v} = \mathbf{A}^+\mathbf{v} + \mathbf{A}^-\mathbf{v} \leq \mathbf{A}^+\mathbf{v}^+ + \mathbf{A}^-\mathbf{v}^- \leq \mathbf{b}.$$

Therefore, for any $\mathbf{v}^- \leq \mathbf{v} \leq \mathbf{v}^+$, $\mathbf{v}$ is a solution to $\mathbf{A}\mathbf{x} \leq \mathbf{b}$. Hence, $I_{\mathbf{x}} = [\mathbf{v}^-, \mathbf{v}^+]$ is an interval solution.

By Proposition 3, we can now find the flexibility $\mathit{flex}(S)$ of a system S of linear constraints by solving the following Linear Program (LP).

$$\text{max} \qquad \sum_i^n (\mathbf{x}^+)_i - \sum_i^n (\mathbf{x}^-)_i \tag{8}$$

$$\text{subject to:} \qquad \mathbf{x}^- \leq \mathbf{x}^+ \tag{9}$$

$$\mathbf{A}^+\mathbf{x}^+ + \mathbf{A}^-\mathbf{x}^- \leq \mathbf{b} \tag{10}$$

Example 5 Consider again the STN S_2 presented in Example 4. Solving the LP for S_2 results in the following solution.

$$\mathbf{x}^- = \begin{bmatrix} 5 \\ 5 \\ 0 \end{bmatrix}, \quad \mathbf{x}^+ = \begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix}. \tag{11}$$

The resulting flexibility $\mathit{flex}(S_2) = \sum_i^n x_i^+ - \sum_i^n x_i^- = 5$. Note that this solution is not unique, although the resulting value for $\mathit{flex}(S_2)$ is. If we consider the STN S_1 it is not difficult to see that $\mathit{flex}(S_1) = 15$. Hence, our flexibility metric avoids the over estimation caused by the slack-based metric flex_s.

As can be seen from this example, over estimation of the available flexibility in linear constraint systems and especially in STPs is avoided by using our metric. The main reason of course is that in our metric every set of values chosen in the flexibility intervals $[x_i^-, x_i^+]$ associated with the temporal variables x_i will result in a valid schedule. This property was missing in the slack-based metrics discussed before: selecting arbitrary values in 'slack' intervals might easiliy result in infeasible schedules.

5 Decoupling Without Loss of Flexibility

Sometimes scheduling problems are specified in a distributive scheduling context (see [1,4,19,20,5]. More specifically, suppose that the events in a scheduling instance are distributed over k independent agents $A_1, \ldots, A_k$, such that each agent is responsible for scheduling a disjoint subset of events. Being an autonomous scheduler, each agent A_i wants to determine its own schedule σ_i and we have to ensure that no matter what choices the individual agents make in composing their individual schedule, *all* original constraints are satisfied. That is, the merge $\sigma = \bigcup_{i=1}^k \sigma_i$ of the individual schedules should always constitute a valid schedule to the original problem.

This problem is known as the so-called *Temporal Decoupling Problem*[2]. Hunsberger [1,2] approached this problem for STPs by making a distinction between *intra-agent constraints*, involving events belonging to a single agent and *inter-agent constraints*, which involve events controlled by different agents. He then proposed a temporal decoupling algorithm that, by tightening intra-agent constraints makes the inter-agent constraints superfluous and returns a set of decoupled subsystems. Each combination of solutions to these subsystems can be merged into a solution of the original problem.

Hunsberger observed that in most cases an arbitrary decoupling is not what we want, since the added (tightened) constraints may limit the flexibilities of the individual subproblems. Hence, because of the decoupling constraints, the sum $\sum_{i=1}^{k} \mathit{flex}(S_i)$ of the flexibilities of the subsystems S_i could be considerably less than the flexibility $\mathit{flex}(S)$ of the original system. He therefore proposed an algorithm that ensures a locally optimal decoupling, i.e. a decoupling such that no individual constraint can be loosened without violating the property of inducing a decoupling. Later on, in [3], the notion of an optimal temporal decoupling was proposed, showing that any linear metric for decoupling could be optimised in linear time. However, that paper did not propose a concrete suggestion for a suitable metric.

Ever since Hunsberger introduced his idea of decoupling, it has been a popular belief in the research community that an *optimal decoupling* i.e. a decoupling that would maximize the flexibilities of the subsystems, would still suffer from some flexibility loss. For example, several experiments have been performed to measure this loss of flexibility [1,4,19]. However, the metrics used were based on the flex_s metric (or variants there-of) we discussed above.

Surprisingly, it turns out that, at least with respect to the flexibility metric we proposed above, this belief is false. In fact, in this section we will show that the rather complicated procedure Hunsberger and others applied to find a decoupling is unnecessary if one uses the flexibility metric we propose: Just computing the flexibility metric gives a decoupling of the system (almost) for free. But not only is a decoupling implied, such a decoupling can even be shown to *preserve* the original flexibility: The sum $\sum_{i=1}^{k} \mathit{flex}(S_i)$ of the flexibilities of the decoupled subsystems S_i equals the flexibility $\mathit{flex}(S)$ of the original system.

Moreover, while Hunsberger showed that STPs can be decoupled efficiently, we show that decoupling can also be done in general systems of linear constraints. So in this section we prove our results about a flexibility preserving (temporal) decoupling method for general linear constraint systems.

5.1 Decoupling in Linear Constraint Systems

To model the idea of decoupling for linear constraint systems, we assume that the variables $\mathbf{x}$ occurring in a (consistent) linear constraint system $\mathbf{Ax} \leq \mathbf{b}$ are decomposed into a set of k vectors $\mathbf{y_i}$ of variables such that $[\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_k}]^t = \mathbf{x}$.

The decoupling problem for linear constraint systems is then to find k matrices $\mathbf{A}_i$ and k vectors $\mathbf{b}_i$ such that

1. for $i = 1, 2 \ldots, k$, S_i is a system $\mathbf{A}_i\mathbf{y}_i \leq \mathbf{b}_i$ and
2. whenever $\sigma_i(\mathbf{y}_i) = \mathbf{v}_i$ is a solution to $\mathbf{A}_i\mathbf{y}_i \leq \mathbf{b}_i$, the schedule σ defined by $\sigma(\mathbf{x})^t = [\sigma(\mathbf{v_1}), \ldots, \sigma(\mathbf{v_k})]^t$ is a solution to $\mathbf{Ax} \leq \mathbf{b}$.

The following result[8] shows that there exists a decoupling where the matrices $\mathbf{A_i}$ can be directly derived from $\mathbf{A}$ and it holds that $\sum_{i=1}^{k} \mathbf{b}_i \leq \mathbf{b}$.

Proposition 4 *Let $\mathbf{Ax} \leq \mathbf{b}$ be a consistent constraint system and $[\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_k}]$ be a decomposition of $\mathbf{x}$. Let $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k$ be chosen such that $\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A_k}]$ and let $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b_k}$ be chosen such that $\sum_{i=1}^{k} \mathbf{b}_i \leq \mathbf{b}$. Then the set $\{\mathbf{A}_i\mathbf{y_i} \leq \mathbf{b}_i\}_{i=1}^{k}$ of constraint systems is a decoupling of $\mathbf{Ax} \leq \mathbf{b}$ if each of the systems $\mathbf{A}_i\mathbf{y_i} \leq \mathbf{b}_i$ is consistent.*

Proof Suppose $\mathbf{A}_i\mathbf{y_i} \leq \mathbf{b}_i$ (for $i = 1, \ldots, k$) are consistent systems, and let $\sigma(\mathbf{y}_i) = \mathbf{v}_i$ (for $i = 1, \ldots, k$) be arbitrary solutions to these systems. Let $\mathbf{v} = [\mathbf{v_1}, \ldots, \mathbf{v_k}]^t$. Then we have

$$\mathbf{Av} = [\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A_k}][\mathbf{v_1}, \ldots, \mathbf{v_k}]^t \leq \sum_{i=1}^{k} \mathbf{b_i} \leq \mathbf{b}$$

Hence, $\mathbf{v}$ is a solution for $\mathbf{Ax} \leq \mathbf{b}$.

Example 6 Consider again the system S_2 and its linear constraint representation given in Example 4. Suppose that $\mathbf{x} = [x_1, x_2, x_3]^t$ is decomposed into $\mathbf{y}_1 = [x_1, x_2]^t$ and $\mathbf{y}_2 = [x_3]$. Using the interval solution presented in Example 5, the following systems constitute a decoupling of the original system $\mathbf{Ax} \leq \mathbf{b}$.

$$\mathbf{A_1} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 1 \\ 0 & -1 \\ -1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ -1 & 0 \\ 0 & -1 \\ 0 & 0 \end{bmatrix}, \ \mathbf{y}_1 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \ \mathbf{b_1} = \begin{bmatrix} 0 \\ 5 \\ 5 \\ 0 \\ -5 \\ -5 \\ 5 \\ 5 \\ 0 \\ -5 \\ -5 \\ 0 \end{bmatrix}; \quad \mathbf{A_2} = \begin{bmatrix} 0 \\ -1 \\ -1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ -1 \end{bmatrix}, \ \mathbf{y}_2 = \begin{bmatrix} x_3 \end{bmatrix}, \ \mathbf{b}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 5 \\ 5 \\ 0 \\ 0 \\ 5 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Note that $\mathbf{b_1} + \mathbf{b_2} \leq \mathbf{b}$.

To achieve an actual decoupling for $\mathbf{Ax} \leq \mathbf{b}$, we now use an interval solution $I_{\mathbf{X}}$ for $\mathbf{Ax} \leq \mathbf{b}$ to compute the vectors $\mathbf{b}_i$ used in the subsystems $\mathbf{A_i}\mathbf{y}_i \leq \mathbf{b}_i$. The idea is as follows:

Let c_i denote the i-th constraint in the constraint system $\mathbf{Ax} \leq \mathbf{b}$. So c_i is

$$a_{i,1}x_1 + \ldots + a_{i,n_1}x_{n_1} + a_{i,n_1+1}x_{n_1+1} + \ldots + a_{i,n_{k-1}+1}x_{n_{k-1}+1} + \ldots + a_{i,n}x_n \leq (\mathbf{b})_i = b_i.$$

Here, the variables in the j-th block $\mathbf{y}_j$ are $x_{n_{j-1}+1}, \ldots, x_{n_j}$.

Following Hunsberger, we distinguish intra- and inter-agent constraints.

[8] This approach can also be seen as a refinement of the general idea of a compact split as defined in [21].

1. An intra-agent constraint is any constraint c_i such that all variables x_h associated with non-zero coefficients $a_{i,h}$ belong to a single block $\mathbf{y}_j$, for example the first constraint in $\mathbf{A}$. That is, agent j is able to set the variables controlled by him to satisfy the constraint. In that case, $(\mathbf{b}_j)_i$ is set to b_i and all remaining entries $(\mathbf{b}_j)'_i$ are set to 0.
2. If c_i contains variables x_h and $x_{h'}$ that belong to different blocks of variables $\mathbf{y}_j$ and are associated with non-zero coefficients, the constraint is said to be an *inter-agent* constraint. In that case, we have to pay attention to all the values $(\mathbf{b}_j)_i$ making sure that their sum does not exceed $(\mathbf{b})_i = b_i$. Therefore, we take an interval solution for $\mathbf{Ax} \leq \mathbf{b}$. By Proposition 3, we know that

   $$a_{i,1}v'_1 + \ldots + a_{i,n_1}v'_{n_1} + a_{i,n_1+1}v'_{n_1+1} + \ldots + a_{i,n_{k-1}+1}v'_{n_{k-1}+1} + \ldots + a_{i,n}v'_n \leq b_i,$$

   where $v'_h = v_h^+$ if $a_{i,h} > 0$ and $v'_h = v_h^-$ if $a_{i,h} \leq 0$. Now let the value $(\mathbf{b}_j)_i$ be equal to the sum of terms belonging to the variables occurring in $\mathbf{y}_j$, i.e., let

   $$(\mathbf{b}_j)_i = a_{i,n_{j_1}+1}v'_{n_{j-1}+1} + \ldots + a_{i,n_j}v'_{n_j}.$$

   This clearly implies that $\sum_{j=1}^{k}(\mathbf{b_j})_i \leq (\mathbf{b})_i = b_i$. Hence, the constraint c_i is implied by k intra-agent constraints $a_{i,n_{j_1}+1}x_{n_{j-1}+1} + \ldots + a_{i,n_j}x_{n_j} \leq (\mathbf{b_j})_i$.

Summarizing, it is now easy to show that the decomposition specified above results in a decoupling of the original system $\mathbf{Ax} \leq \mathbf{b}$.

Proposition 5 *Let* $\mathbf{Ax} \leq \mathbf{b}$ *be a constraint system,* $[\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_k}]$ *a decomposition of* $\mathbf{x}$ *and let* $\{\mathbf{A_j y_j} \leq \mathbf{b_j} : j = 1, \ldots, k\}$ *be a set of systems as defined above. Then* $\{\mathbf{A_j y_j} \leq \mathbf{b_j} : j = 1, \ldots k\}$ *is a decoupling of* $\mathbf{Ax} \leq \mathbf{b}$.

Proof Suppose, on the contrary, that the merge σ of some combination of individual schedules σ_i, violates $\mathbf{Ax} \leq \mathbf{b}$. Without loss of generality, assume that $(\mathbf{A})_i\sigma(\mathbf{x}) > b_i$. By the definition of a decoupling, such a violation is only possible if the elements of $X(\mathbf{a_i})$ belong to $m \geq 2$ vectors $\mathbf{y_{j_1}}, \ldots, \mathbf{y_{j_m}}$. Now observe that for every $1 \leq j \leq m$, we have $(A_j)_i\sigma(\mathbf{y_j}) \leq (\mathbf{b_j})_i$. Hence, by the definition of the matrices $\mathbf{A}_j$,

$$\mathbf{A}\sigma(\mathbf{x}) = [\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k][\sigma_1(\mathbf{y_1}), \sigma_2(\mathbf{y_2}), \ldots, \sigma_k(\mathbf{y_k}]^t) = \sum_{i=1}^{k} \mathbf{A}_i\sigma_i(\mathbf{y_i}) \leq \sum_{j=1}^{k} \mathbf{b}_j \leq \mathbf{b}$$

contradicting the assumption that the merge of the schedules is not a assignment that satisfies the original system.

This shows that our decomposition method results in a valid decoupling of a linear constraint system. Moreover, it is easy to see that given the interval solution $I_X = [\mathbf{v}^-, \mathbf{v}^+]$ the computations involved can be performed in linear time.

5.2 Flexibility preservation

Intuitively, one would expect decoupling to adversely affect the flexibility of a system. Let us define the flexibility of a decoupled system $\{S_i\}_{i=1}^k$ as $\sum_{i=1}^k \mathit{flex}(S_i)$. We want to compute the ratio $\sum_{i=1}^k \mathit{flex}(S_i)/\mathit{flex}(S)$, which will be smaller than or equal to 1, because we need more constraints to ensure that the merge of every combination of

solutions of the parts will be a solution of the original system, and adding constraints can only decrease flexibility.

It may come as a surprise that using the flexibility metric we have defined in the previous section and the decomposition method we have discussed, we can prove that decoupling of linear constraint systems does not result in any loss of flexibility:

Proposition 6 *Let S be a linear constraint system $\mathbf{Ax} \leq \mathbf{b}$, $[\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_k}]$ a decomposition of $\mathbf{x}$ and let $\{S_i : i = 1, \ldots, k\}$ be a decoupled set of linear systems $\{\mathbf{A_j y_j} \leq \mathbf{b_j} : j = 1, \ldots, k\}$ using an interval solution I_X. Then we have $\mathit{flex}(S) = \sum_{i=1}^{k} \mathit{flex}(S_i)$ whenever I_X is an interval solution ensuring maximal flexibility for S.*

Proof It is easy to see that any decoupling method will ensure $\sum_{i=1}^{k} \mathit{flex}(S_i) \leq \mathit{flex}(S)$. Given $\mathbf{Ax} \leq \mathbf{b}$, let $I_X = [\mathbf{x}^+, \mathbf{x}^-]$ be an interval solution to S ensuring maximal flexibility.

For every subsystem S_j of the form $\mathbf{A_j y_j} \leq \mathbf{b_j}$ consider the set $I_{\mathbf{y_j}}$, being the restriction of $I_{\mathbf{X}}$ to $\mathbf{y_j}$. We show that $I_{\mathbf{y_j}} = [\mathbf{v_j}^-, \mathbf{v_j}^+]$ is an interval solution for S_j. By Proposition 3 it suffices to show that $\mathbf{A}_j{}^+\mathbf{v}^+ + \mathbf{A}_j{}^-\mathbf{v}^- \leq \mathbf{b}_j$. Take any row i from A_j. Then we have to show that

$$a_{i,n_{j-1}+1} v'_{n_{j-1}+1} + \ldots + a_{i,n_{j-1}+2} v'_{n_{j-1}+2} + \ldots + a_{i,n_j} v'_{n_j} \leq (\mathbf{b}_j)_i,$$

where $v'_h = v_h^+$ if $a_{i,h} > 0$ and $v'_h = v_h^-$ if $a_{i,h} \leq 0$. We distinguish the following cases:

1. all entries in the i-th row of $\mathbf{A}_j$ are 0. In that case the inequality holds.
2. all non-zero entries in the i-th row of $\mathbf{A}$ are associated with variables belonging to $\mathbf{y}_j$, that is c_i is an intra-agent constraint. In that case $(\mathbf{b}_j)_i = (\mathbf{b})_i$ and the inequality holds.
3. c_i is an inter-agent constraint. In that case, $(\mathbf{b}_j)_i = \sum_{h=n_{j-1}+1}^{n_j} a_{i,h} v'_h$ and the inequality also holds.

Since the decompositions $\mathbf{y}_j$ do not overlap, the sum of the flexibilities of all subsystems S_i now equals

$$\sum_{j=1}^{k} \mathit{flex}(S_j) = \sum_{j=1}^{k} \sum_{h=1}^{n_j} (\mathbf{v}^+)_h - (\mathbf{v}^-)_h = \sum_{i=1}^{n} (\mathbf{v}^+)_i - (\mathbf{v}^-)_i = \mathit{flex}(S)$$

Hence, there exists at least one decoupling realizing a total flexibility equal to the original flexibility $\mathit{flex}(S)$.

Example 7 Consider Example 6. The decoupling is based on the interval solution presented in Example 5. The interval solution $I_{y_1} = [(5,5)^t, (5,5)^t]$ and the interval solution $I_{y_2} = [(0), (5)]$. The sum of the flexibilities of the two subsystems (5) equals the flexibility of the original system (5).

Example 8 As a simple example, assume that the tasks from Example 1 are distributed over two agents, such that one agent owns x_1, x_2 and x_3, and the other agent owns x_4 and x_5. Notice that one of the additional constraints introduced in Example 2 to enforce resource consistence connects the two agents. In Figure 5 a decomposition is shown: x_1 and x_5 are additionally constrained (in bold), such that the constraint connecting x_1 and x_5 can be removed (dotted). Again, the sum (10) of the flexibilities of both agents equals the flexibility of the original system (10).

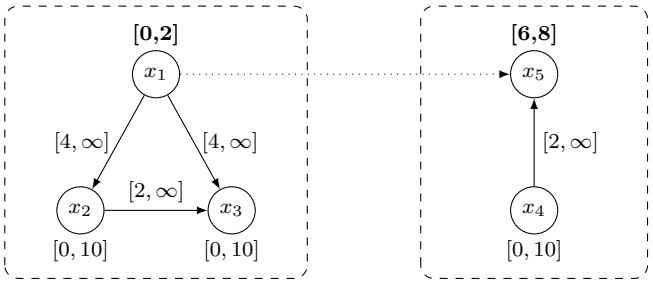


Fig. 5 A decomposition of the STN in Figure 3, assuming that x_1, x_2 and x_3 belong to one agent, and x_4 and x_5 to another agent.

6 Conclusion and Discussion

In this paper we concentrated on flexibility of the Resource Constrained Project Scheduling Problem (RCPSP) by investigating flexibility metrics for Simple Temporal Problems. These STPs could be obtained by reducing an RCPSP instance to this purely temporal constraint specification. Instead of encoding a single solution, the obtained STP representation encodes a set of solutions to the original RCPSP-instance. After reviewing some existing flexibility metrics we concluded that these fail to capture the correlation between events specified in the STP, resulting in an overestimation of the available flexibility in the system. We proposed an intuitively more acceptable new flexibility metric for general linear constraint systems and also showed that this flexibility metric can be used in autonomous scheduling: Whenever a scheduling problem has to be solved by more than one independent actor, we would like to let these actors solve their part of the problem independently without violating the original constraints. This is the so-called *decoupling problem*. As a byproduct of the flexibility computation, we get a decoupling of the instance almost for free: for every possible partitioning of the event space among k agents, a decoupling can be computed in linear-time. Even more importantly, we show that—contrary to popular belief—such a decomposition does not need to affect the flexibility of the original linear constraint system.

There are a couple of questions we have to answer in future research. First of all, there are quite some heuristic methods available for constraint posting. We would like to investigate a number of them using our flexibility metric. An interesting question in this context is that various constant posting techniques are using some kind of slack-based heuristic to make a choice about what precedence constraint to post. Using our metric we hope to show that some of these heuristics might be improved.

Secondly, we observe that while the slack-based heuristics overestimate flexibility, our metric could be accused of underestimating it. While our metric can be seen as a metric of concurrent flexibility, we could also envision a metric that using some partial order between the events takes into account the influence of this order on the flexibility. The computation of such flexible metrics will require some multi-level optimization methods and the complexity involved is not completely clear at this moment.

Thirdly, note that using our metric the total flexibility of a system can be obtained in many, sometimes even quite unfair, ways. For example, preserving the total flexibility of the system could mean that a decoupling heavily favors some specific agents by giving them a lot of flexibility, while harming other agents by taking away their flexibility. To prevent this, we need to come up with fair flexibility distribution schemes as an

objective to grant every agent an almost equal portion of flexibility. We could propose more *fair* distributions of flexibility and evaluate their performance by comparing it to the maximally obtainable flexibility as we have proposed in our paper to establish the *price of fairness*.

Finally, another question we did not deal with in this paper is how we should distribute flexibility over the events in order to ensure that a given interval schedule as proposed in this paper could enhance the robustness of a schedule. Research in this direction also should pay attention to the dependencies of events upon other events (as expressed by the STP) to minimize scheduling violations. Together with stochastic extensions of the STP this would enable stochastic interpretations of the total flexibility of an STP and encourage empirical research into optimal distributions of flexibility over the events in order to ensure maximal (expected) robustness of the resulting schedule.

References

1. Hunsberger, L.: Group decision making and temporal reasoning. Ph.D. thesis, Harvard University, Cambridge, Massachusetts (2002)
2. Hunsberger, L.: Algorithms for a temporal decoupling problem in multi-agent planning. In: Proceedings AAAI (2002)
3. Planken, L., de Weerdt, M., Witteveen, C.: Optimal temporal decoupling in multi-agent systems. In: Proceedings AAMAS (2010)
4. Boerkoel, J., Durfee, E.: Distributed algorithms for solving the multi-agent temporal decoupling problem. In: Proceedings AAMAS (2011)
5. Brambilla, A.: Artificial Intelligence in Space Systems: Coordination Through Problem Decoupling in Multi-Agent Planning for Space Systems. Lambert Acad. Publishing (2010)
6. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artificial Intelligence **49** (1991)
7. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. European J. Operational Research **207**(1) (2010)
8. Demeulemeester, E.L., Herroelen, W.S.: New benchmark results for the resource-constrained project scheduling problem. Management Science **43**(11) (1997)
9. Schutt, A., Feydy, T., Stuckey, P.J., Wallace, M.G.: Solving RCPSP/max by lazy clause generation. J. Scheduling **16**(3) (2013)
10. Policella, N., Wang, X., Smith, S., Oddi, A.: Exploiting temporal flexibility to obtain high quality schedules. In: Proceedings AAAI (2005)
11. Policella, N., Cesta, A., Oddi, A., Smith, S.: Solve-and-robustify: Synthesizing partial order schedules by chaining. J. Scheduling **12**, 299–314 (2009)
12. Oddi, A., Smith, S.F.: Stochastic procedures for generating feasible schedules. In: Proceedings AAAI (1997)
13. Baar, T., Brucker, P., Knust, S.: Tabu search algorithms and lower bounds for the RCPSP. In: S. Voß, et al. (eds.) Meta-Heuristics. Springer (1999)
14. Policella, N., Cesta, A., Oddi, A., Smith, S.: From precedence constraint posting to partial order schedules: A CSP approach to robust scheduling. AI Communications **20**(3) (2007)
15. Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)
16. Floyd, R.: Algorithm 97: Shortest path. Communications of the ACM **5**(6) (1962)
17. Chtourou, H., Haouari, M.: A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. Computers & Industrial Engineering **55**(1) (2008)
18. Policella, N., Smith, S.F., Cesta, A., Oddi, A.: Generating robust schedules through temporal flexibility. In: Proceedings ICAPS (2004)
19. Boerkoel, J., Durfee, E.: A distributed approach to summarizing spaces of multi-agent schedules. In: Proceedings AAAI (2012)
20. Boerkoel, J., Durfee, E.: Distributed reasoning for multi-agent simple temporal problems. Journal of AI Research **47** (2013)
21. Brodsky, A., Kerschberg, L., Varas, S.: Optimal constraint decomposition for distributed databases. In: Proceedings ASIAN (2004)

Abstracts

MISTA 2013

Optimal Parallel Systems for Random Scheduling Times

Syouji Nakamura · Mingchin Chen · Keiko Nakayama

Abstract There are few researches dealt with scheduling problems with failures of system operation. This paper proposes optimaization problems of a parallel system in which how many number of units should be provided for a job with random working times. First, an optimal number of units for a job with a single working time is obtained. Next, when the system operates for a job with n works, optimal numbers when n is constant and random are obtained, respectively. When failure and working times are exponential, optimal policies are discussed analytically and computed numerically. Finally, intoroducing excess and shortage costs, all results are rewritten similarly and newly.

1 Introduction

Suppose that a system operates for a job with random working times. In general, it would be difficult to predict exactly the job completion time. So that, the scheduling problem is an important concern in production management. A general overview of stochastic scheduling problems were be done [1,2]. However, scheduling models with failures of system operation have received few attention. The scheduling problem with n jobs on a single machine subjected failures was examined [3].

Intoroducing excess and shortage costs, several problems how to determine the schedule time were solved [4, p.82],[5]. This paper proposes optimization problems of a parallel system in which how many number of units should be provided for a job with random working time [6]. For example, we can give an actual exsample as one of applications, that an equipment rental office for construction should prepare for the number of machines needed on the demand.

Syouji Nakamura
Kinjo Gakuin University, Japan
E-mail: snakam@kinjo-u.ac.jp

Mingchin Chen
Fu-Jen Catholic University, Taiwan
E-mail: 081438@mail.fju.edu.tw

Keiko Nakayama
Chukyo University, Japan
E-mail: nakayama@mecl.chukyo-u.ac.jp

At first, we obtain an optimal number N^* of units of a parallel system for a job with constant and random working times, respectively. Next, when a paralled system operates for a job with n works, we obtain optimal numbers N^* when n is constant and random. When both failure time and working time are exponential, optimal numbers which minimize the expected costs are discussed analytically and computed numerically. Finally, intoroducing excess and shortage costs, optimal policies are shown to be rewritten similarly.

2 n Working Times

Consider the scheduling problems for a job with n works ($n = 1, 2, \dots$). It is assumed that $S_j (j = 1, 2, \dots, n)$ are the working times for a job and independent and have an identical distribution $\Pr\{S_j \le t\} = W(t)$ with finite mean $1/\omega$. Then, the probability that n works finish until time t is

$$\Pr\{S_1 + S_2 + \cdots + S_n\} = W^{(n)}(t),$$

where $\phi^{(j)}(t)(j = 1, 2, \dots)$ denotes the j-fold Stielties convolution of any function $\phi(t)$ and $\phi^{(0)}(t) \equiv 1$ for $t \ge 0$.

The total expected cost until the n work completion is

$$C_1(N; n) = c_2 + (c_1 - c_2) \int_0^\infty [1 - W^{(n)}(t)] \mathrm{d}F(t)^N + c_0 N \quad (N = 0, 1, 2, \dots). \quad (1)$$

From the inequality $C_1(N + 1; n) - C_1(N; n) \ge 0$,

$$\int_0^\infty \overline{F}(t) F(t)^N \mathrm{d}W^{(n)}(t) \le \frac{c_0}{c_1 - c_2}. \quad (2)$$

Thus, there exists a finite and unique minimum $N_1^* (0 \le N_1^* < \infty)$ which satisfies (2).

In particuler, when $F(t) = 1 - \mathrm{e}^{-\lambda t}$ and $W(t) = 1 - \mathrm{e}^{-\omega t}$, (2) is

$$\sum_{j=0}^{N} \binom{N}{j} (-1)^j \left[\frac{\omega}{\omega + (j+1)\lambda}\right]^n \le \frac{c_0}{c_1 - c_2}, \quad (3)$$

whose left-hand side decreases with N from $[\omega/(\omega + \lambda)]^n$ to 0. Therefore, if $[\omega/(\omega + \lambda)]^n > c_0/(c_1 - c_2)$ then there exists a finite and unique minimum $N_1^* (1 \le N_1^* < \infty)$ which satisfies (3).

Table 2 presents optimal N_1^* for $c_0/(c_1 - c_2)$ and n when $\lambda/\omega = 1$, and indicates that N_1^* increases with $c_0/(c_1 - c_2)$, however, do not always increase with n.

Next, the system operates for a job with works which arrive at a counter appeared in repairman problems [7, p.139]. In this case, the number of works is not constant, however, it may be estimated statistically.

It is assumed that the number n of works is a random variable with a discrete probability function $p_k \equiv \Pr\{n = k\} (k = 1, 2, \dots)$ and $\sum_{k=1}^\infty p_k = 1$. Then, the total expected cost until the work completion is

$$C_1(N; p) = c_2 + (c_1 - c_2) \sum_{k=1}^{\infty} p_k \int_0^\infty [1 - W^{(k)}(t)] \mathrm{d}F(t)^N + c_0 N \quad (N = 0, 1, 2, \dots). \quad (4)$$

Table 1 Optimal N_1^* for n works when $\lambda/\omega = 1$

	n			
$\frac{c_0}{c_1-c_2}$	2	3	4	5
0.5	0	0	0	0
0.2	1	0	0	0
0.1	2	1	0	0
0.05	4	4	2	0
0.02	9	10	8	4
0.01	14	18	18	14

From the inequality $C_1(N+1;p) - C_1(N;p) \geq 0$,

$$\sum_{k=1}^{\infty} p_k \int_0^{\infty} \overline{F}(t)F(t)^N \mathrm{d}W^{(k)}(t) \leq \frac{c_0}{c_1 - c_2}. \tag{5}$$

Thus, there exists a finite and unique minimum $N_1^*(0 \leq N_1^* < \infty)$ which satisfies (5).

In particular, suppose that $p_k = (\theta^{k-1}/(k-1)!)\mathrm{e}^{-\theta}(k = 1, 2, \dots)$ with mean $\theta + 1$. Then, when $F(t) = 1 - \mathrm{e}^{-\lambda t}$ and $W(t) = 1 - \mathrm{e}^{-\omega t}$,

$$\sum_{j=0}^{N} \binom{N}{j} (-1)^j \left[\frac{\omega}{\omega + (j+1)\lambda}\right] \exp\left[-\frac{(j+1)\lambda\theta}{\omega + (j+1)\lambda}\right] \leq \frac{c_0}{c_1 - c_2}, \tag{6}$$

whose left-hand side decreases with N from

$$\frac{\omega}{\omega + \lambda} \exp\left[-\frac{\lambda\theta}{\omega + \lambda}\right] \tag{7}$$

to 0. Therefore, if (7) is greater than $c_0/(c_1 - c_2)$ then there exists a finite and unique minimum $N_2^*(1 \leq N_2^* < \infty)$ which satisfies (6).

Table 3 presents optimal N_2^* for $c_0/(c_1 - c_2)$ and $\theta + 1$ when $\lambda/\omega = 1$, and indicates that N_2^* are almost the same as N_1^* and show similer tendencies to Table 2.

Table 2 Optimal N_2^* for works with mean $\theta + 1$ when $\lambda/\omega = 1$

	$\theta + 1$			
$\frac{c_0}{c_1-c_2}$	2	3	4	5
0.5	0	0	0	0
0.2	1	0	0	0
0.1	2	1	1	0
0.05	4	3	2	1
0.02	8	8	7	5
0.01	13	15	15	13

3 Conclusions

We have solved analytically and numerically scheduling problems of a parallel system in which how many number of units should be provided for a job with random working times. So that, if the number of works could be estimated statistically, we could derive easily an optimal number of units for a parallel system from these results.

There exist many different kinds of works and a variety of suitable redundant systems for operating such works in practical fields. However, by modifying and extending methods and techniques used in this paper, these results could be applied to actual models. These results would offer interesting topics for formulating scheduling problems with system failures.

Acknowledgements This work is partially supported by the Grant-in-Aid for Scientific Research (C) of Japan Society for the Promotion of Science under Grant No. 22500897 and No. 24530371.

References

1. S.C.Sarin, E.Erel and G.Steiner, Sequencing jobs on a single machine with a common due date and stochastic processing times, European J. Operational Research, 51, 188-198 (1991)
2. M.Pinedo, Scheduling Theory, Algorithm and System. Prentice Hall, Englewood Cliffs, NJ (2008)
3. X.Zhou and X.Cai, General stochastic single-machine scheduling with regular cost function, Mathematical and Computing Modeling, 26, 95-108 (1997)
4. T.Nakagawa, Advanced Reliability Models and Maintenance Policies. Springer, London. (2008)
5. M.Chen and T.Nakagawa, Optimal scheduling of random works with reliability applications, Asia-Pacific J. Operational Research, 29, doi: 10.1142/S0217595912500273 (2012)
6. M.Chen and T.Nakagawa, Optimal redundant systems for works with random processing time, To appear in Reliability Engineeering and System Safety (2013)
7. R.E.Barlow and F.Proschan, Mathematical Theory of Reliability. Wiley, New York (1965)

MISTA 2013

Fast Approximation Schemes for Boolean Programming and Scheduling Problems Related to Positive Convex Half-Product

Hans Kellerer · Vitaly Strusevich

The topic of designing approximation schemes for scheduling problems with min-sum objective functions has recently drawn considerable attention. While for many problems of this range purpose-built approximation schemes have been developed, a general framework has been identified based on reformulation of the original scheduling problems in terms of minimization problems of quadratic Boolean programming. The Half-Product Problem and the closely related Symmetric Quadratic Knapsack Problem appear to be among the most suitable models.

We study a version of the Half-Product Problem and its modification with the knapsack constraint, establishes conditions under which the problems admit fast fully polynomial-time approximation schemes, describes the relevant algorithms and discusses their scheduling applications.

Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be a vector with n Boolean components. Consider the function

$$H(\mathbf{x}) = \sum_{1 \le i < j \le n} \alpha_i \beta_j x_i x_j - \sum_{j=1}^{n} \gamma_j x_j, \tag{1}$$

where for each j, $1 \le j \le n$, the coefficients α_j and β_j are non-negative integers, while γ_j is an integer that can be either negative or positive. This function and the term "half-product" were introduced by Badics and Boros [1], who considered the problem of minimizing the function $H(\mathbf{x})$ with respect to Boolean decision variables with no additional constraints. The function $H(\mathbf{x})$ is called a *half-product* since its quadratic part consists of roughly half of the terms of the product $\left(\sum_{j=1}^{n} \alpha_j x_j\right)\left(\sum_{j=1}^{n} \beta_j x_j\right)$.

We refer to the problem of minimizing function $H(\mathbf{x})$ of the form (1), as *Problem HP*. This problem is NP-hard in the ordinary sense, even if $a_j = b_j$ for all $j = 1, 2, \ldots, n$, as proved in [1]. It has numerous applications, mainly to machine

Hans Kellerer
Institut für Statistik und Operations Research, Universität Graz, Universitätsstraße 15, A-8010, Graz, Austria
E-mail: hans.kellerer@uni-graz.at

Vitaly Strusevich
School of Computing and Mathematical Sciences, University of Greenwich, Old Royal Naval College, Park Row, Greenwich, London SE10 9LS, U.K.
E-mail: V.Strusevich@greenwich.ac.uk

scheduling. Notice that in those applications a scheduling objective function usually is written in the form

$$F(\mathbf{x}) = H(\mathbf{x}) + K, \tag{2}$$

where K is a given additive constant. We refer to the problem of minimizing function $F(\mathbf{x})$ of the form (2), as *Problem HPAdd*.

Consider the function

$$P(\mathbf{x}) = \sum_{1 \le i < j \le n} \alpha_i \beta_j x_i x_j + \sum_{j=1}^{n} \mu_j x_j + \sum_{j=1}^{n} \nu_j \left(1 - x_j\right) + K, \tag{3}$$

where all coefficients α_j are positive integers, and β_j, μ_j, ν_j and K are non-negative integers. Following Janiak et al. [3], we call the problem of minimizing the function $P(\mathbf{x})$ of the form (3) the *Positive Half-Product Problem* or *Problem PosHP*. Notice that Problem PosHP is a special case of Problem HPAdd.

In all Half-Product Problems introduced above, the minimum is sought for over all $n-$dimensional Boolean vectors, i.e., they are quadratic Boolean programming problems with no additional constraints. We also study a more restricted version of Problem PosHP, in which an additional knapsack constraint is introduced, i.e., the problem

$$\begin{aligned} \text{Minimize} \quad & P(\mathbf{x}) = \sum_{1 \le i < j \le n} \alpha_i \beta_j x_i x_j + \sum_{j=1}^{n} \mu_j x_j + \sum_{j=1}^{n} \nu_j \left(1 - x_j\right) + K \\ \text{Subject to} \quad & \sum_{j=1}^{n} \alpha_j x_j \le A \\ & x_j \in \{0,1\},\ j = 1, 2, \ldots, n, \end{aligned} \tag{4}$$

which we call the *Positive Half-Product Knapsack Problem* and denote as *Problem PosHPK*.

Similarly to the classical knapsack problem Problem PosHPK contains a linear *knapsack* constraint. We can view the value α_j as the weight of item j, $1 \le j \le n$, i.e., $x_j = 1$ means that item j is placed into a knapsack with capacity A, while $x_j = 0$ means that the corresponding item is not placed into the knapsack.

An important feature of our problem is that the coefficients α_j in the knapsack constraint are the same as in the quadratic terms of the objective function. The latter feature makes Problem PosHPK to be a special case of another quadratic knapsack problem, namely the problem

$$\begin{aligned} \text{Minimize} \quad & S(\mathbf{x}) = \sum_{1 \le i < j \le n} \alpha_i \beta_j x_i x_j + \sum_{1 \le i < j \le n} \alpha_i \beta_j (1 - x_i)(1 - x_j) \\ & + \sum_{j=1}^{n} \mu_j x_j + \sum_{j=1}^{n} \nu_j (1 - x_j) + K \\ \text{Subject to} \quad & \sum_{j=1}^{n} \alpha_j x_j \le A \\ & x_j \in \{0,1\},\ j = 1, 2, \ldots, n. \end{aligned} \tag{5}$$

Following [4], we call the latter problem the *Symmetric Quadratic Knapsack Problem*, or *Problem SQK*. A comprehensive review of the results on Problem SQK and its scheduling applications is given by Kellerer and Strusevich [6].

Badics and Boros [1] give the first FPTAS for Problem HP but its running time is $O(n^2 \log \sum \alpha_j/\varepsilon)$ is not strongly polynomial. The first FPTAS for Problem HP that requires strongly polynomial time $O(n^2/\varepsilon)$ is due to Erel and Ghosh [2].

In all known applications of quadratic Boolean programming problems related to the Half-Product the objective function is convex. That is why we study Problem PosHPK and its relaxed version, Problem PosHP without a knapsack constraint, provided that the objective function $P(\mathbf{x})$ is convex. For each of these problems with a convex objective function, this paper delivers an FPTAS that requires $O\left(n^2/\varepsilon\right)$ time. This running time is much smaller than $O\left(n^4/\varepsilon^2\right)$ established by [5] for Problem SQK (under additional assumptions that include the convexity of an objective function) or $O\left(n^4 \log\log n + n^4/\varepsilon^2\right)$ provided by Xu [7] for an arbitrary Problem SQK. The developed FPTAS can be applied to several scheduling problems, resulting in improved approximation schemes for their solution, namely: minimizing the total weighted flow time with controllable processing times, minimizing the makespan with controllable release dates, minimizing the total weighted flow time for two models of scheduling with rejection.

References

1. Badics, T., & Boros, E., Minimization of half-products, Mathematics of Operations Research, 33, 649–660 (1998).
2. Erel, E., & Ghosh, J.B., FPTAS for half-products minimization with scheduling applications, Discrete Applied Mathematics, 156, 3046–3056 (2008).
3. Janiak, A., Kovalyov, M.Y., Kubiak W., & Werner, F., Positive half-products and scheduling with controllable processing times, European Journal of Operational Research, 165, 416–422 (2005).
4. Kellerer, H., & Strusevich, V.A., Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. Algorithmica, 57, 769–795 (2010).
5. Kellerer, H., & Strusevich, V.A., Minimizing total weighted earliness-tardiness on a single machine around a small common due date: an FPTAS using quadratic knapsack, International Journal of Foundations of Computer Science, 21, 357–383 (2010).
6. Kellerer, H., & Strusevich, V.A., The symmetric quadratic knapsack problem: approximation and scheduling applications. 4OR – A Quarterly Journal of Operations Research, 10, 111-161 (2012).
7. Xu, Z., A strongly polynomial FPTAS for the symmetric quadratic knapsack problem, European Journal of Operational Research 218, 377–381 (2012).

MISTA 2013

A resource loading problem in the aircraft line maintenance industry

Jorne Van den Bergh · Jeroen Beliën · Roel Leus · Fabrice Talla Nobibon

1 Introduction

This paper studies a resource loading problem with timing decisions, situated in the aircraft line maintenance industry. Aircraft line maintenance is the routine maintenance between flight arrival and departure at an airport. For each job a time window is specified within which the maintenance must be performed, bounded by the scheduled time of arrival (STA) and the scheduled time of departure (STD) of the aircraft. Furthermore, each job is characterized by a maintenance workload, expressed in person-hours. The maintenance time is inversely proportional to the number of workers assigned to the job. Thus, if a job is executed by, say, two persons, the job lasts only half the time it would take if it were done by only one person. These intensities are not required to be fixed during the job execution but can change monotonously over time. This means that a job is executed either with non-decreasing or with non-increasing intensities. For instance, one cannot start a job with one person, then continue with two persons and finish again with one person. This monotonicity constraint implies that the jobs are executed without preemption.

Our work belongs to tactical planning in capacity management, which is concerned with allocating sufficient resources to deal with the incoming demand, as effectively and profitably as possible. Hans [3] defines the resource loading problem (RLP) as scheduling a given set of tasks such that the cost of overtime is minimized and all due

Jorne Van den Bergh
Center for Informatics, Modeling and Simulation, HUBrussel (Belgium)
Affiliated researcher Research Center for Operations Management, KU Leuven (Belgium)
E-mail: Jorne.VandenBergh@hubrussel.be

Jeroen Beliën
Center for Informatics, Modeling and Simulation, HUBrussel (Belgium)
Affiliated researcher Research Center for Operations Management, KU Leuven (Belgium)
E-mail: Jeroen.Belien@hubrussel.be

Roel Leus
ORSTAT, KU Leuven (Belgium)
E-mail: Roel.Leus@kuleuven.be

Fabrice Talla Nobibon
ORSTAT, KU Leuven (Belgium)
E-mail: Fabrice.TallaNobibon@kuleuven.be

dates are respected, or the cost of due-date violation is minimized within the available capacity. Our model is related to the basic models, that can be found in the theses of De Boer [2] and Hans [3]. There are some distinguishing characteristics, however. Firstly, our model takes into account monotonicity and non-preemption. Secondly, the algorithms proposed by Hans [3] are developed for the *time driven* RLP, for which the goal is to meet all deadlines, in such a way, that the usage of nonregular capacity is minimized. We do not restrict ourselves to the subproblem in which the jobs are not allowed to be tardy. All jobs are characterized by a release time, due time and deadline. Thirdly, we do not consider precedence constraints between the jobs. The most recent related work in tactical planning was done by Wullink et al. [10] and Wullink [11], whose major contribution is the incorporation of uncertainty (under the form of a number of discrete scenarios). In this paper we do not incorporate uncertainty explicitly, but the tardiness penalties in the objective do allow to protect from variability in some of the parameters.

In the operational scheduling literature, models quite similar to our planning of work packages have been studied under the title of project scheduling with 'variable-intensity' or 'continuously divisible' activities. This work seems to have been initiated by Weglarz [9], who considered a continuous time horizon, and was followed by Leachman [7], Leachman et al. [8], Kogan and Shtub [6] and Kis [4,5]. The papers that are most relevant to this article are those that use a bucketized time horizon and integer-programming (IP) methods, which are Hans [3] (branch-and-price algorithm), Kis [4, 5] (branch-and-cut), and Alfieri et al. [1] (IP).

2 Problem statement

The planning horizon is discretized into H periods; period t ($t = 1, \ldots, H$) runs from time (instant) $t-1$ to t (period 1, for instance, is the interval $[0;1]$). Our planning problem takes as input a workforce roster that defines the capacity for each period, which changes over time. We are given a set $J = \{1, 2, \ldots, n\}$ of jobs, which require one resource type with regular capacity μ_t in period t ($t = 1, \ldots, H$). To ensure feasibility, nonregular capacity o_t ($t = 1, \ldots, H$) is allowed at a cost C^o per period. Each job j has a *release time* r_j before which it cannot be started, which equals its STA and which is a time instant (note that the earliest period of execution for job j is period $r_j + 1$). The STD of each job j is the desired completion time or *due time* d_j for job j. Each job j has a hard deadline $\bar{d}_j \geq d_j$, which cannot be exceeded. If job j is executed after its due time d_j, a penalty of C_j^d per tardy period is assigned. We represent job j's work content by W_j (expressed in man-periods). Each job j has a pre-specified minimum work content L_j that must be executed and a maximum work content U_j that can be processed during each period in which it is scheduled (i.e., $L_j, U_j \in \mathbb{R}^+ (j \in J)$).

There are three decisions to be made: for each job in which (consecutive) periods it should be executed and in which proportion, and in which periods extra (nonregular) capacity should be added. We propose the following linear formulation for the problem. We use binary decision variables $\delta_j \in \{0, 1\}$ ($j \in J$), which are 1, respectively 0, if the intensity of the execution of job j is non-decreasing, respectively non-increasing, with time over the consecutive periods with non-zero workload. Further, let $x_{jt} \in \{0, 1\}$ ($j \in J$, $t \in \{r_j + 1, \ldots, \bar{d}_j\}$) be a binary variable, equal to 1 if job j is produced in period t, and 0 otherwise, and $\chi_{jt} \in \{0, 1\}$ is 1 when job j has period t as last period of execution, 0 otherwise. Finally, the continuous variable $y_{jt} \in [0; U_j]$ ($j \in J$,

$t \in \{r_j + 1, \ldots, \bar{d}_j\}$) represents the workload of job j executed in period t. A linear formulation of the problem with these variables is

$$(F1) \qquad \text{minimise} \qquad \sum_{j=1}^{n} C_j^d \left(\sum_{t=d_j+1}^{\bar{d}_j} (t - d_j)\chi_{jt} \right) + \sum_{t=1}^{T} C^o o_t \tag{1}$$

subject to

$$\sum_{t=r_j+1}^{\bar{d}_j} y_{jt} = W_j \qquad j = 1, \ldots, n \tag{2}$$

$$L_j x_{jt} \leq y_{jt} \leq U_j x_{jt} \qquad j = 1, \ldots, n \quad t = r_j + 1, \ldots, \bar{d}_j \tag{3}$$

$$x_{j,t-1} \leq x_{jt} + \sum_{k=r_j+1}^{t-1} y_{jk}/W_j \qquad j = 1, \ldots, n \quad t = r_j + 2, \ldots, \bar{d}_j \tag{4}$$

$$y_{j,t-1}/W_j \leq y_{jt}/W_j + (1 - \delta_j) + (1 - x_{jt}) \qquad j = 1, \ldots, n \quad t = r_j + 2, \ldots, \bar{d}_j \tag{5}$$

$$y_{jt}/W_j \leq y_{j,t-1}/W_j + \delta_j + (1 - x_{j,t-1}) \qquad j = 1, \ldots, n \quad t = r_j + 2, \ldots, \bar{d}_j \tag{6}$$

$$x_{jt} - x_{j,t+1} \leq \chi_{jt} \qquad j = 1, \ldots, n \quad t = r_j + 1, \ldots, \bar{d}_j \tag{7}$$

$$\sum_{t=r_j+1}^{\bar{d}_j} \chi_{jt} = 1 \qquad j = 1, \ldots, n \tag{8}$$

$$\sum_{j=1}^{n} y_{jt} \leq \mu_t + o_t \qquad t = 1, \ldots, T \tag{9}$$

The objective function (1) minimises the total cost. Constraint set (2) guarantees that each job is completely maintained. Constraint set (3) makes sure that the number of resources assigned to a job lies within a specified interval for those periods during which it is executed. Non-preemption is ensured with constraint set (4). Constraint sets (5) and (6) are the monotonicity constraints. Constraint sets (7)-(8) fix the χ_{jt}-variables, which indicate the last period in which processing takes place. Finally, constraint set (9) ensures that in every time period, the number of allocated resources does not outnumber the available capacity (i.e., regular and nonregular capacity).

3 Solution approach

The formulation ($F1$) is our first 'compact' formulation, in which the number of decision variables is polynomially bounded. The quality of various compact formulations will be empirically verified. ($F1$) can for instance be extended with the following constraints as valid inequalities:

$$\left\lceil \frac{W_j}{U_j} \right\rceil \leq \sum_{t=r_j+1}^{\bar{d}_j} x_{jt} \leq \left\lfloor \frac{W_j}{L_j} \right\rfloor \qquad j = 1, \ldots, n \tag{10}$$

These constraints state upper and lower bounds on the number of (consecutive) periods in which each job will be scheduled for execution (hence, on its duration).

An alternative for the non-preemption constraint (4) is the following:

$$x_{j,t+1} - x_{jt} + x_{j,\ell+1} - x_{j\ell} \leq 1 \quad j = 1, \ldots, n \quad t, \ell = r_j + 1, \ldots, \bar{d}_j - 1 \quad t < \ell \quad (11)$$

This constraint states that no two periods t, l can be labeled 'starting periods', i.e., with production in $t+1$ but not in t and idem for l.

The second type of formulations under consideration, are entitled 'extensive' formulations and these have an exponential number of variables, which is why a branch-and-price algorithm will be used to solve them. These extensive formulations are retrieved by reformulating the compact formulations by means of Dantzig-Wolfe (DW) decomposition on the x-variables. Hereby, the columns represent so-called 'execution masks' for jobs, which indicate for each period whether a job should be processed (within the minimal and maximal proportions) or not.

References

1. Alfieri A., Tolio T. and Urgo M. (2011). A project scheduling approach to production planning with feeding precedence relations. *International Journal of Production Research* 49(4), 995–1020.
2. De Boer R. (1998). Resource-Constrained Multi-Project Management - A Hierarchical Decision Support System. Ph.D. thesis, University Of Twente, The Netherlands.
3. Hans E.W. (2001). Resource Loading by Branch-and-Price Techniques. Ph.D. thesis, University of Twente, The Netherlands.
4. Kis T. (2005). A branch-and-cut algorithm for scheduling of projects with variable-intensity activities. *Mathematical Programming*, 103, 515–539.
5. Kis T. (2006). RCPS with variable intensity activities and feeding precedence constraints. Chapter 5 in: Jozefowska J. and Weglarz J. *Perspectives in Modern Project Scheduling*. International Series in Operations Research & Management Science, Volume 92, Springer.
6. Kogan K. and Shtub A. (1999). Scheduling projects with variable-intensity activities: The case of dynamic earliness and tardiness costs. *European Journal of Operational Research* 118, 65–80.
7. Leachman R.C. (1983). Multiple resource leveling in construction systems through variation of activity intensities. *Naval Research Logistics Quarterly*, 30(2), 187–198.
8. Leachman R.C., Dincerler A. and Kim S. (1990). Resource-constrained scheduling of projects with variable-intensity activities. *IIE Transactions*, 22(1), 31–39.
9. Weglarz J. (1981). Project scheduling with continuously-divisible, doubly constrained resources. *Management Science*, 27(9), 1040–1053.
10. Wullink G., Gademann A.J.R.M., Hans E.W. and van Harten A. (2004). Scenario-based approach for flexible resource loading under uncertainty. *International Journal of Production Research*, 42(24), 5079–5098.
11. Wullink G. (2005) Resource Loading Under Uncertainty. Ph.D. thesis, University of Twente, The Netherlands.

MISTA 2013

Scheduling chains with constant timelags and variable release times on a single machine

Gulcin Ermis · Ceyda Oguz

1 Introduction

We study scheduling chains with constant timelags and variable release times on a single machine which is denoted by $1|chains(l); p_i = p; r_i|C_{max}$ and has an open complexity status. The problem is to schedule n identical tasks with constant processing times, $p_i = p$, release times, r_i, and chain precedence relations together with constant timelag constraint, l, on a single machine. Timelag constraint requires delaying the start time of a task if it is scheduled immediately after its predecessor. Simultaneous processing of tasks on the machine and preemption are not allowed. The aim of the scheduling problem is to minimize the makespan, C_{max}. In this study we assume that $l < p$ and develop an O(n^2) algorithm to obtain the optimal solution.

2 Proposed method

The proposed algorithm schedules tasks iteratively by taking the precedence graph and the release time values into consideration. At every iteration, the release time values and the number of remaining tasks in different chains are updated based on the scheduled task of the previous iteration. Release time update considers the completion time of already scheduled tasks, timelag restrictions and the original release time values. A decision point further in time is also determined to be able to make a decision at that iteration so that global optimum can be reached. Without determination of this decision point, the algorithm would work as a greedy algorithm. Then, based on certain conditions, a priority is calculated for every candidate task for scheduling at that iteration. Considering the decision point and the priority of candidate tasks, a selection is made among the candidate tasks and the selected task is scheduled. The selection is made based on several rules, which were developed in this study. These rules will generate a group of alternative assignment orders of the tasks. Then the task

Gulcin Ermis
Koc University
E-mail: gulermis@ku.edu.tr

Ceyda Oguz
Koc University
E-mail: coguz@ku.edu.tr

that leads to the construction of the best partial assignment order till the decision point is scheduled at the current position of the sequence. Once a task is scheduled, the iteration is completed and the whole procedure is repeated until all tasks are scheduled.

Due to the page limit we present below one case which leads to determine the decision point so that a task can be scheduled at a given iteration:

Let K be the total number of chains and n_k be the number of tasks in chain k where $k = 0, \ldots, K-1$. $r_{k,i}$ denotes the release time and $p_{k,i} = p$ denotes the processing time of task i, $i = 0, \ldots, n_k - 1$, in chain k, $k = 0, \ldots, K-1$. We define the chains α, β, θ and ζ which contain the tasks with the following release times, respectively: $r_{\alpha,0} = min_{0 \leq k \leq K-1} r_{k,0}$, $r_{\beta,0} = min_{0 \leq k \leq K-1} r_{k \neq \alpha,0}$, $r_{\theta,0} = min_{0 \leq k \leq K-1} r_{(k \neq \alpha \& k \neq \beta),0}$ and $r_{\zeta,0} = min_{0 \leq k \leq K-1} r_{(k \neq \alpha \& k \neq \beta \& k \neq \theta),0}$.

Proposition: If the following conditions are met, the optimal schedule starts at time $r_{\beta,0}$.

(i) $r_{\beta,0} - r_{\alpha,0} < l$,
(ii) There exists a task g such that it is the first task in chain α for which $2gp + l \leq r_{\alpha,g} - r_{\alpha,0} < (2g+1)p + l$ holds,
(iii) $r_{\alpha,i+1} - r_{\alpha,0} \leq (2i+2)p$ for all $i = 0, \ldots, g-2$,
(iv) $r_{\beta,i+1} - r_{\beta,0} \leq (2i+2)p$ for all $i = 0, \ldots, g-2$,
(v) $r_{\theta,0} \geq r_{\alpha,0} + 2gp + l$,
(vi) $r_{\beta,g} - r_{\beta,0} < 2gp + l$,
(vii) $r_{\beta,g} < r_{\alpha,g}$,
(viii) $a \neq 0$ where a is a counter of the following inequality till it becomes invalid: $r_{\theta,0} > start^{\beta}_{i+1} = max(start^{\beta}_{i} + 2p, r_{\beta,i+1})$ for $i = g, \ldots, n_\beta - 2$ and $start^{\beta}_{g} = max(r_{\beta,0} + 2gp, r_{\beta,g})$,
(ix) $b = a + 1$ where b is a counter of the following inequality: $min(r_{\theta,0}, r_{\beta,i+1}) < start^{\alpha}_{i+1} = max(start^{\alpha}_{i} + 2p, r_{\alpha,i+1}) < start^{\alpha}_{i} + 2p + l$ for $i = g, \ldots, g+a$ and $start^{\alpha}_{g} = max(r_{\alpha,0} + 2gp, r_{\alpha,g})$,
(x) $max_{g \leq i \leq g+a-1} |(start^{\alpha}_{i+1} + 2p) - (start^{\beta}_{i+1} + 2p)| < l$,
(xi) $min(r_{\beta,g+a+1}, r_{\theta,1}, r_{\zeta,0}, max[max(start^{\beta}_{g+a} + 2p, r_{\theta,0}) + 2p + l; r_{\alpha,g+a+2})]) < start^{\alpha}_{g+a+1} + 2p$.

Proof: Suppose there are K chains in accordance with the conditions given in the Proposition. For a contradiction, assume that optimal schedule starts at time $r_{\alpha,0}$. Then, there exists a partial sequence σ^1 with

$$r_{\alpha,0} + (2g + 2a + 5)p + l \leq C_{max}(\sigma^1) < r_{\alpha,0} + (2g + 2a + 6)p + (a+2)l.$$

By swapping the task 0 of chain α with the task 0 of chain β in sequence σ^1 and assigning the remaining tasks accordingly, a new sequence σ^2 starting at time $r_{\beta,0}$ and with

$$r_{\beta,0} + (2g + 2a + 5)p \leq C_{max}(\sigma^2) < r_{\alpha,0} + (2g + 2a + 6)p + (a+2)l$$

is found. Although the upper bounds are equal, the smallest possible value of $C_{max}(\sigma^2)$ is less than the smallest possible value of $C_{max}(\sigma^1)$ since

$$r_{\beta,0} + (2g + 2a + 5)p < r_{\alpha,0} + (2g + 2a + 5)p + l$$

due to the inequality $r_{\beta,0} - r_{\alpha,0} < l$. Therefore, initiating the schedule at time $r_{\alpha,0}$ by task 0 of chain α is not optimal. The proof is valid for any value of g and a. □

There exist other conditions which are valid when one or more inequalities defined for the given proposition are violated. Those conditions are grouped under different propositions where the appropriate strategy is proposed for determining the task and the start time of the task that will be assigned into the schedule at the first iteration. For the following iterations the same conditions and rules apply in the general manner, however some modifications are needed to take into consideration the completion time of the assigned task and its effect on the sequence due to timelag restrictions. Modified conditions are summarized below.

At any iteration except for the first one, completion time of the task that was assigned in the previous iteration is found. For the tasks which are allowed to be assigned into the current position of the schedule, earliest times they can start are determined. Earliest start times are the maximum of the completion time of the previously assigned task and the release time of the candidate task, where the candidates belong to the chains different than the one whose task was assigned into previous position. Otherwise, when the candidate task belongs to the same chain, earliest start time is calculated by taking the maximum of completion time of the previously assigned task plus timelag and release time of the unassigned task with the highest priority in the chain. Earliest start times are used instead of $r_{\alpha,0}$, $r_{\beta,0}$, $r_{\theta,0}$ and $r_{\zeta,0}$.

MISTA 2013

Minimizing the dissemination length in the one-datum-unit data transfer problem

Ronan Bocquillon · Antoine Jouglet

1 Introduction

Systems of systems (SoS) are collections of systems which interact between themselves to achieve functions most of which require the *transfer* of information through the whole SoS. Elements of a mobile-interacting-system set can profit in an opportunistic manner from *contacts*, arising when two entities are close enough, to share data. Such exchanges allow them to collaborate to route information from a subset of sources to a subset of recipient systems. This collaboration becomes for instance necessary when contact durations are relatively short with respect to the size of information to be disseminated. This information is then split and spread, passing eventually through non recipient *messenger* systems whose role is to carry and later forward data. Existing works already deal with such environments in both opportunistic and *delay-tolerant networks* [2] fields. Most of the time, no assumption is made regarding the contacts happening between the different entities making up the SoS, although realistic *reliable predictions* can be made about contacts for many real applications, *e.g.* satellites networks (as the trajectory of each system follows well known laws from dynamics).

We thus study a problem aiming at profiting from knowledge about collaboration possibilities [1,3] to route information from its sources towards its destinations before the end of a considered time horizon: the general question to be answered is what piece of information should be transferred from which system to which other one and when (during which contact) ? In this paper, we are investigating a data transfer optimization problem in a simple case where systems can only exchange a given amount of data during each contact. The aim is to find a transfer plan minimizing the dissemination length, *i.e.* a transfer plan from which all recipients get the information as soon as possible. The remainder of the paper is organized as follows. Section 2 first formally describes this problem. In Section 3, we introduce some dominance rules and techniques to improve solving performances. We also provide a few computational results to show the efficiency of these methods.

Ronan Bocquillon · Antoine Jouglet
Heudiasyc, UMR CNRS 7253, Université de Technologie de Compiègne
E-mail: {ronan.bocquillon, antoine.jouglet}@hds.utc.fr

2 Problem definition

We consider a set $\mathcal{S} = \{S_1, \ldots, S_q\}$ of q interacting (mobile) systems and a unique datum $D = \{1, \ldots, u\}$ of u datum units. Each of these units represents an indivisible and unitary datum fragment (in bytes for instance). Each system $S_l \in \mathcal{S}$ initially owns a subset $\mathcal{O}_l \subseteq D$ of datum units (making data sources up). The subset $\mathcal{R} \subseteq \mathcal{S}$ defines the systems wishing to retrieve D (recipients) before a given time horizon.

To enable the dissemination of D, systems can exchange datum units whenever they are close enough to communicate between themselves. Such an opportunity is called a *contact* in the following. We here suppose the contacts are perfectly known or easily predictable at any time. We thus define a contacts sequence $\sigma = \{\sigma_1, \ldots, \sigma_m\}$ of m ordered pairs of $\mathcal{S}^2$. During contact $(s, r) \in \sigma$, receiving system r can receive from sending system s at most one datum unit $k \in D$ that s already owns (from the start or thanks to previous contacts). From this time, r is also supposed to store datum unit k in its internal memory. In the following, s_c and r_c refer respectively to the sender and the receiver of contact $\sigma_c \in \sigma$, i.e $\sigma_c = (s_c, r_c)$.

A transfer plan is a function $\phi : \{1, \ldots, m\} \rightarrow \{\emptyset, \{1\}, \{2\}, \ldots, \{u\}\}$ where $\phi(c)$ designates the datum unit received by r_c during σ_c. $\phi(c) = \emptyset$ if nothing is transferred during σ_c. Given a transfer plan ϕ, we associate with each system $S_l \in \mathcal{S}$, a set of states O_l^t defined by: $\forall c \in \{1, \ldots, m\},\ O_{r_c}^c = O_{r_c}^{c-1} \cup \phi(c),\ O_l^c = O_l^{c-1}\ \forall l \neq r_c$ with initial set $O_l^0 = \mathcal{O}_l$. Set O_l^t contains every datum units retrieved by S_l after the t first contacts of σ. A transfer plan ϕ is said to be valid if a sender always transfers a datum unit it already owns, *i.e.* $\phi(c) \in \{\emptyset\} \cup \{\{k\}\ /k \in O_{s_c}^{c-1}\}\ \forall c \in \{1, \ldots, m\}$. Finding a valid transfer plan was proved to be strongly NP-Hard.

With any valid transfer plan ϕ, we associate the delivery length $\lambda_l(\phi)$ of a recipient system $S_l \in \mathcal{R}$ such that $\lambda_l(\phi) = \min\{t / O_l^t = D\}$. It corresponds to the smallest contact index t after which system S_l owns every datum units making up D. If such an index does not exist, we assume in the following that $\lambda_l(\phi) = \infty$. We also associate with ϕ, its dissemination length $\lambda(\phi) = \max_{S_l \in \mathcal{R}} \lambda_l(\phi)$. It corresponds to the smallest index t at which every recipients is served. In this paper, we tackle the problem which attempts to find a valid transfer plan minimizing $\lambda(\phi)$.

3 Solving techniques

We investigated several integer linear programming and constraint programming based methods to solve the problem. It appears that solvers can be challenged, even for small instances. In this talk, we present several dominance rules allowing the search space of the problem to be reduced.

The transfer occurring at time c with respect to transfer plan ϕ is said to be null if and only if no datum unit is transmitted throughout contact σ_c, *i.e.* $\phi(c) = \emptyset$. The transfer occurring at time c with respect to transfer plan ϕ is said to be improving if and only if the receiver gets a new datum unit during σ_c, *i.e.* $|O_{r_c}^{c-1}| < |O_{r_c}^c|$. The set of transfer plans in which all transfers are null or improving is dominant. Such transfer plans are said to be *minimal*.

Any transfer plan ϕ is said to be *active* if there is no contact $\sigma_c \in \sigma$ during which a datum unit $k \in D$ is transmitted from $s_c \in \mathcal{S}$ to $r_c \in \mathcal{S}$ while it could have been done earlier by exploiting a non-improving transfer $\phi(c')$ (see equation below). The set of active transfer plans is also dominant.

$$\forall c \in \{2,\dots,m\}, \phi(c) = \{k\}, |O^{c-1}_{r_c}| < |O^{c}_{r_c}|,$$
$$\nexists c' \in \{1,\dots,c-1\}, r_{c'} = r_c, k \in O^{c'-1}_{s_{c'}}, |O^{c'-1}_{r_{c'}}| = |O^{c'}_{r_{c'}}|$$

On the top of these dominance rules, we designed some algorithms allowing some additional constraints to be deduced for a given instance. We designed for example a few propagation algorithms which show that some contacts are fruitless in all dominant solutions. The corresponding transfers $\phi(c)$ are thus fixed to $\emptyset$. In some cases, we can also decide the datum unit which is necessarily transferred.

These techniques have already been tested as preprocessings and dynamical deduction rules. To illustrate the efficiency of our methods, we provide below some results obtained for a set of instances when our deduction techniques are applied before the solving of an integer linear program modeling the problem or/and dynamically applied during its resolution. For different pairs (q, u), where q and u are the number of systems and datum units respectively, we generated 10 instances. m is the average number of contacts over the instances. These latter were each solved using `C++` and `CPLEX 12.5` over an Intel I7 at 2.60GHz with 8Go RAM (running on only one thread). We provide results obtained by `CPLEX` alone (*cf.* ILP), by using our techniques as preprocessings before running `CPLEX` (*cf.* PRE), and finally by applying our algorithms both before and throughout (dynamically) `CPLEX` process (*cf.* ALL). To evaluate performances, we compare the average rate P of contacts which survived to preprocessings, the average rate F of transfers which were proved to be non-null in all dominant solutions and the average number D of variables which were dynamically fixed in addition to those chosen by `CPLEX` during the branching step. We also consider the average number N of nodes processed during the branch-and-cut algorithm and the average computational time T in seconds (including preprocessings) required to solve each instance to optimality.

			ILP		**PRE**				**ALL**		
q	u	m	N	T	P	F	N	T	D	N	T
10	3	125	1832	0.65	46%	12%	10	0.04	55	8	0.03
20	3	371	1883	2.01	49%	5%	18	0.13	68	31	0.15
20	4	290	11828	15.4	62%	11%	50	0.30	248	25	0.31
50	4	718	6560	18.75	77%	7%	630	2.63	572	33	2.08
50	5	718	48885	293	85%	8%	1356	5.07	2610	120	6.73

4 Conclusion

In this paper, we proposed a new approach to study routing issues in systems of systems whose topology is dynamical but predictable. We provided a formalization and some resolution techniques to tackle one simple case. Note that we now plan to investigate robust optimization and situations where contact durations are uncertain.

Acknowledgements These works are financed by the *Conseil Régional de Picardie*, and are supported by the *Labex MS2T* (we refer to http://www.utc.fr/labexms2t/ for details).

References

1. J. Alonso and K. Fall. A linear programming formulation of flows over time with piecewise constant capacity and transit times. Technical report, Intel Research, Berkeley, 2003.
2. K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '03*, page 27, New York, USA, 2003. ACM Press.
3. S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '04*, page 145, New York, USA, 2004. ACM Press.

MISTA 2013

A large neighborhood search for the Shift Design Personnel Task Scheduling Problem with Equity

Tanguy Lapègue, Odile Bellenguez-Morineau, Damien Prot

Keywords Fixed Job Scheduling Problem, Personnel Task Scheduling Problem, Large Neighborhood Search, Constraint Programming

1 Introduction

This extended abstract deals with a specific scheduling problem arising in a company specialized in drug evaluation and pharmacology research. The aim is to build employee timetables covering the demand given by a set of fixed tasks while maximizing the equity among workers. We propose a Large Neighborhood Search to solve it. This method is based on the CP approach presented in [1]. The remainder of the abstract is organized as follows: Section 2 provides a description of the industrial problem and Section 3 is devoted to the proposed LNS.

2 Problem description

The Shift Design Personnel Task Scheduling Problem with Equity (SDPTSP-E cf. [1]) consists in building fine rosters respecting legal and organizational constraints while covering the demand, given by a set of fixed tasks which have to be assigned to qualified and available employees. Consequently, while designing employee rosters, it is required to assign tasks to employees, and if possible, in a fair way. This problem arises in a company which carries out clinical studies on behalf of pharmaceutical laboratories which deliver to the company a very specific protocol containing a description of all the clinical tasks to be performed, along with their relative starting time and duration. Each week, the various tasks related to the ongoing protocols have to be assigned to qualified and available employees. The individual rosters resulting from this assignment have to respect a set of organizational and legal constraints, and they are expected to be as fair as possible. However, employees do not work only on clinical tasks, but also

LUNAM Université, École des Mines de Nantes, IRCCyN UMR CNRS 6597, (Institut de Recherche en Communications et Cybernétique de Nantes), 4, rue Alfred Kastler - B.P. 20722, 44307 Nantes Cedex 3, France, E-mail: {tanguy.lapegue, odile.morineau, damien.prot}@mines-nantes.fr

on administrative tasks, which do not require any specific skills and can be done whenever during the week. Consequently, these administrative tasks are not scheduled and employees are free to work on them whenever they want, provided they are not already assigned to a clinical task in the mean time. To be sure that employees have enough time to work on their administrative tasks, the chief nurse defines a clinical targeted working time for each employee. When designing the planning, the chief nurse tries for each employee to reach his/her clinical targeted working time. Consequently, the objective is to minimize the gap between the employee working the most compared to his/her target, and the one working the less, compared to his/her target. For instance, if each employee is one hour above his/her clinical targeted working time, the solution is optimal, and in practice, it means that the additional workload has been shared among employees. In order to build a feasible planning, several constraints have to be considered. The most important ones are:

1. tasks cannot be preempted;
2. each task must be assigned to one employee;
3. employees cannot be assigned to tasks they are not skilled or available for;
4. working days must last less than 10h;
5. working days must be followed by a resting time of at least 11h;
6. a weekly break of at least 35h must occur in the planning of each employee.

A solution respecting the constraint (2) corresponds to a complete assignment, as opposed to a partial one. However, some instances do not admit a complete assignment, meaning that the regular workforce has to be strengthened with externals. In this case, the chief nurse has to estimate the number of externals, which may be difficult. Consequently, we relax the constraint (2), so that the chief nurse may get some insight on the need of externals by looking at a partial assignment. This is handled by a soft constraint in order to favor complete assignments to the detriment of partial ones.

The SDPTSP-E is related to many problems such as the Fixed Job Scheduling Problem, the Interval Scheduling Problem, the Ground Crew Rostering Problem, the Personnel Task Scheduling Problem and the Nurse Rostering Problem. A detailed literature review, given in [1] explains the main differences between these problems and the SDPTSP-E.

3 Proposed Large Neighborhood Search

The use of LNS is motivated by the results obtained in [1]. This approach, based on pure CP, is able to find good solutions on small instances but encountered some difficulties to find complete assignments on bigger instances. However, when a complete assignment is not found, the number of unassigned tasks is on average very small. Moreover, the default exploration of the search tree turned out to be quite ineffective. Using LNS allows the search to focus more intensively on interesting parts of the search space, while reducing the size of the problem handled by the CP solver.

Once an initial solution has been obtained, which is done in our case with the CP approach, the general idea of LNS is to iterate on a destruction and construction loop in order to improve the current solution. During the destruction step, a part of the current solution is destroyed, leading to a neighborhood, i.e. a set of potential solutions close to the current one. The construction step consists in exploring the resulting search space

to find a new solution. It is commonly admitted that neighborhoods should focus on related variables in order to be able to reach new solutions ([3]). Since the SDPTSP-E is composed of an assignment problem along with a shift design problem, several neighborhoods may be considered. In this extended abstract we consider two kinds of neighborhoods:

1. **Time**: select randomly an initial task and unassign the corresponding set of overlapping tasks. When working on a partial assignment, this neighborhood selects preferably an unassigned task as the initial one.
2. **Personnel**: select randomly a subset of employees and unassign their set of tasks. When working on a complete assignment, this neighborhood selects preferably workers who are the furthest from their targeted clinical working time.

Moreover, we also increase the size of the neighborhood after a given number of consecutive non improving iterations. Thus, small improvements can be achieved quickly by exploring relatively small neighborhoods, giving more time to explore bigger neighborhoods. Once the neighborhood is defined, a CP approach based on [1] runs for a few seconds to find an improving solution. Because of the size of the neighborhoods and the choice of the construction method, neighborhoods are not solved to optimality. Consequently, further thoughts must be given to the branching and exploration strategies: since an important part of the solution is already frozen, guiding the search is not as important as it is when building a solution from scratch. Consequently, the construction step relies highly on random branching and exploration, combined with a restart criterion. However, we also use some of the deterministic strategies presented in [1]: some of them focus on finding a complete assignment whereas others focus on improving the equity. Thus, each neighborhood is explored by several strategies, and the local best solution is used in the next iteration.

The last step of the proposed approach is a local search that tries to improve the equity. It starts by reducing (respectively increasing) the workload of workers who work too much (respectively not enough) regarding to their targeted workload. This is done first by moving tasks from one worker to another and then by swapping tasks of different durations. The post optimization keeps running as long as it is able to improve the equity.

In order to validate this approach, extensive tests are carried out on 720 instances, in order to compare the results obtained with the pure CP method and the LNS-CP method presented in this extended abstract. A detailed description of these instances along with the corresponding benchmark is available on [2]. Results will be presented at the conference, but the first ones are promising.

References

1. Tanguy Lapègue, Damien Prot, and Odile Bellenguez-Morineau. A constraint-based approach for the shift design personnel task scheduling problem with equity. *Computers & Operations Research*, 40(10):2450–2465, 2013.
2. Damien Prot, Tanguy Lapègue, and Odile Bellenguez-Morineau. Personnel Task Scheduling Problem Library (*https://sites.google.com/site/ptsplib/*), 2013.
3. Paul Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Principles and Practice of Constraint Programming (CP 1998), volume 1520 of LNCS*, pages 417–431, 1998.

MISTA 2013

Order acceptance and scheduling problem with time-dependent revenues

Chen Cheng • Renjie He • Feng Yao • Zhenyu Yang

1 Introduction

The order acceptance and scheduling problem has been considered as an important topic in make-to-order systems over the past two decades. Order acceptance decides which orders are to be accepted from all of the orders which can not be totally filled, while scheduling fixes on the start time for processing each of the accepted orders. Diversity problems that combine the decisions of selection and scheduling have been studied by both researches as well as practitioners. Slotnick[1] summarized variants of order acceptance and scheduling problems under different settings and objectives from various application areas. Our motivation originates from the agile earth observing satellite (AEOS) scheduling problems. Compared with the traditional non-agile earth observing satellite, AEOS is mobile along three axes (roll, pitch and yaw). The mobility considerably increases the length of time windows for observing targets because the satellite can look forward or look afterward with large pitch angles. However, relatively larger pitch angles may result in the deterioration of image qualities. The revenue or user satisfaction obtained by imaging a target depends on the starting time (or end time) of imaging with respect to the pitch angle. Thus, joint decisions that combine deciding which targets are to be imaged and fixing the appropriate start time of imaging have to be made. We formulate the problem as order acceptance and scheduling problem with time-dependent revenues on a single machine environment. Hall[2] incorporated Lagrangean relaxation into a dynamic programming algorithm without taking into consideration tardiness penalty and sequence dependent setup times. Charnsirisakskul[3] studied a single machine production system where setup times were negligible and proposed a mixed-integer programming method. Oguz[4] studied order acceptance and scheduling problem on a single machine environment with tardiness penalty and sequence-dependent setup times developed a mixed-integer linear programming method, which can solve problems with 15 orders to optimality. Cesaret[5] developed a tabu search algorithm supported with a probabilistic local search procedure for order acceptance and scheduling problem with an objective of

Chen Cheng
School of Information System and Management, National University of Defense Technology, Changsha, Hunan Province, 410073, P.R.China.
E-mail: chencheng.prc@gmail.com

RenJie He
School of Information System and Management, National University of Defense Technology, Changsha, Hunan Province, 410073, P.R.China.
E-mail: renjiehe@nudt.edu.cn

Feng Yao
School of Information System and Management, National University of Defense Technology, Changsha, Hunan Province, 410073, P.R.China.
E-mail: fengyao@nudt.edu.cn

Zhenyu Yang
School of Information System and Management, National University of Defense Technology, Changsha, Hunan Province, 410073, P.R.China.
E-mail: zhyuyang@ieee.org

maximizing total net revenue minus tardiness penalty. Lin[6] studied exactly the same problem and developed an artificial bee colony algorithm combined with an iterated greedy heuristic.

This study aims to investigate order acceptance and scheduling problem with time-dependent revenues. To be more precisely, we introduce user-preferred earliest end time for each order. Besides the tardiness penalty, an order is also penalized if the end time of the processing of the order is earlier than the given user-preferred earliest end time. This new problem arises from the application of AEOS scheduling.

2 Problem definition

There are n incoming orders without any precedence constraints to be processed on a single machine. The machine can process at most one order at any time and preemption is not allowed. Each order i is released at time r_i, after which the order is available. Processing time p_i is needed to process the order. If the end time for processing the order is later than user-preferred earliest end time f_i and earlier than user-preferred latest end time l_i, a revenue e_i is obtained. However, if the end time for processing the order is earlier than f_i or later than l_i, a penalty on the revenue is incurred. If the completion time of order i is later than the deadline d_i, we say the order is not accepted and no revenue is gained from the order. A sequence-dependent setup time s_{ij} is incurred if order j is processed immediately after order i in the processing sequence. It is not necessarily that s_{ij} is equal to s_{ji}. The objective is to find a subset of accepted orders that maximize the total net revenue obtained. The problem we study is a generalization of the problem studied in [4, 5], which was probed to be NP-hard.

3 Solution method

Considering the complexity of the problem, it seems to be more appropriate to use heuristic methods. The chemical reaction optimization of Lam[7] is a recently developed meta-heuristic which mimics the interactions of molecules in a chemical reaction to reach a low energy stable state. This method has been tested without problem specific adaptations on some combinatorial optimization problems like quadratic assignment problem and resource-constrained project scheduling problem. These experiments proved the high efficiency of the chemical reaction optimization method. We design new operators for order acceptance and scheduling problem with time-dependent revenues and propose a local search procedure for the problem.

The algorithm is implemented using C# programming language in Windows 7 environment. Computational experiments provide a comparative assessment of the employed method against the ILOG OPL in terms of their solution quality under given computation time. If the processing time of each order is relatively long, the results of the employed chemical reaction optimization algorithm are considerably better than that of ILOG OPL.

Acknowledgements This work is supported by the National Natural Science Foundation of China (Grant No. 61203180, No. 71071156 and No. 71201171).

References

1. Susan A. Slotnick. Order acceptance and scheduling: A taxonomy and review [J]. European Journal of Operational Research, 2011, 212: 1-11.

2. Nicholas G. Hall, Michael J. Magazine. Maximizing the value of a space mission [J]. European Journal of Operational Research, 1994, 78: 224-241.
3. Kasarin Charnsirisakskul, Paul M. Griffin, Pinar Keskinocak. Pricing and scheduling decisions with leadtime flexibility [J]. European Journal of Operational Research, 2006, 171: 153-169.
4. Ceyda Oguz, F. Sibel Salman, Zehra Bilginturk Yalcin. Order acceptance and scheduling decisions in make-to-order systems [J]. International Journal of Production Economics, 2010, 125: 200-211.
5. Bahriye Cesaret, Ceyda Oguz, F. Sibel Salman. A tabu search algorithm for order acceptance and scheduling [J]. Computers & Operations Research, 2012, 39: 1197-1205.
6. S-W Lin, K-C Ying. Increasing the total net revenue for single machine order acceptance and scheduling problems using an artificial bee colony algorithm [J]. Journal of the Operational Research Society, 2012, 0 (0): 1-19.
7. Albert Y.S. Lam, Victor O.K. Li. Chemical-Reaction-Inspired Metaheuristic for Optimization [J]. IEEE Transactions on Evolutionary Computation, 2010, 14 (3): 381-399.

MISTA 2013

Scheduling round robin tournaments for amateur sport leagues

Dries R. Goossens · Frits C.R. Spieksma

1 Introduction

The vast majority of the sports scheduling papers deal with professional leagues (see e.g. [1], [2], and [3]). Academic interest in problems faced by amateur leagues is far more rare. Indeed, as far as we are aware, Knust [5], and Schönberger et al. [6] are the only contributors in this area, with a discussion of a scheduling problem faced by a regional non-professional table-tennis federation in Germany.

In general, there are two main differences between professional and amateur sport leagues with respect to scheduling. Firstly, there are usually few spectators, and hence the home advantage is quite limited. Consequently, successions of home or away matches (breaks) are hardly relevant in amateur leagues, whereas in professional leagues, the alternation of home and away matches (break minimization) is usually an important constraint. Secondly, stadium or ground availability is in general more limited, for instance because the team's venue tends to be shared with other teams or sports disciplines. Consequently, more rounds than the minimum number of rounds required to schedule all the matches will be necessary to satisfy these constraints. In other words, amateur leagues use so-called temporally relaxed schedules, as opposed to professional sport leagues, where schedules are usually temporally constrained (see e.g. [4] for football in Europe).

2 Problem description and mathematical formulation

This work deals with round robin tournament scheduling for a set of T amateur teams, with $|T| = n$. In a round robin tournament, each team meets each other team a fixed number of times. The double round robin tournament, where each team meets each

Dries R. Goossens
PostDoc researcher for Research Foundation - Flanders
ORSTAT, Faculty of Business and Economics, KU Leuven
E-mail: Dries.Goossens@kuleuven.be

Frits C.R. Spieksma
ORSTAT, Faculty of Business and Economics, KU Leuven
E-mail: Frits.Spieksma@kuleuven.be

other team twice (once at its home venue, and once at the opponent's venue) is the most common. All games have to be scheduled within the set of time slots (i.e. days) $S = \{1, 2, ..., |S|\}$, ranging from the first day of the competition till the last.

Each team i provides a list of time slots $H_i \subseteq S$ for which their home ground is available. Home games for a team can only be scheduled on time slots from this list. Obviously, each team should at least provide as many time slots as it has opponents, i.e. $|H_i| \geqslant n - 1$. This list is called the *home game set*. Some teams may have a slot on the same weekday every other week; other teams may have a more irregular home game set. Each team can also provide a list of calender days $A_i \subseteq S$ on which it doesn't want to play a match; we call this list the *forbidden game set*. Teams can use this list to avoid matches in the Christmas and New Year period, on holidays, during an exam period, etc. The forbidden game set implies that on all days not in the list, the team is able to play an away game. A team is not allowed to play twice on the same day, or more than twice in a period of R_{max} days. Finally, there should also be at least q calendar days between two matches featuring the same teams.

The goal is to develop a schedule with for each team a balanced spread of their matches over the season. More in particular, teams wish to avoid having two matches in a period of R_{max} days or less. We use p^r to denote the penalty incurred for every pair of matches played by a team within period of $r \in R = \{2, 3, ..., R_{max}\}$ days. Obviously, having 2 matches in 2 days is considered more unpleasant than having 2 matches in 4 days. The main idea behind this, is that most players prefer not to fully spend their weekend with their sport. Moreover, matches packed together could also lead to injuries. If a team has more than R_{max} days between two consecutive matches, we assume that the league organizers no longer care, and consider any number greater than R_{max} as equally adequate.

Our main decision variable is x_{ijs}, which is 1 if team i plays a home game against team $j \neq i$ on slot s, and 0 otherwise. The variable y_{ist} is 1 if team i plays a match on slot s, followed by its next match on slot t, for each s and t such that $t > s$ and $t - s + 1 \leqslant R_{max}$, and 0 otherwise. We can now write the following formulation for our problem:

$$\text{minimize} \sum_{i \in T} \sum_{s \in S} \sum_{t \in S} p^{t-s+1} y_{ist} \tag{1}$$

subject to

$$\sum_{s \in H_i \setminus A_j} x_{ijs} = 1 \qquad \forall i, j \in T \tag{2}$$

$$\sum_{j \in T} (x_{ijs} + x_{jis}) \leqslant 1 \qquad \forall i \in T, s \in S \tag{3}$$

$$y_{ist} \leqslant \sum_{j \in T} (x_{ijs} + x_{jis}) \qquad \forall i \in T, s, t \in S \tag{4}$$

$$y_{ist} \leqslant \sum_{j \in T} (x_{ijt} + x_{jit}) \qquad \forall i \in T, s, t \in S \tag{5}$$

$$\sum_{j \in T} (x_{ijs} + x_{jis}) + \sum_{j \in T} (x_{ijt} + x_{jit}) - \sum_{j \in T} \sum_{k=s+1}^{t-1} (x_{ijk} + x_{jik}) - 1 \leqslant y_{ist} \qquad \forall i \in T, s, t \in S : s < t \tag{6}$$

$$x_{ijs} + x_{jis'} \leqslant 1 \qquad \forall i, j \in T, s \in H_i, s' \in H_j : |s - s'| < q \qquad (7)$$

$$\sum_{j \in T} \sum_{k=s}^{s+R_{max}-1} (x_{ijk} + x_{jik}) \leqslant 2 \qquad \forall i \in T, s \in S \qquad (8)$$

$$x_{ijs} = 0 \qquad \forall i, j, \in T, s \notin H_i \vee s \in A_j \qquad (9)$$

$$x_{ijs} \in \{0, 1\} \qquad \forall i, j, \in T, s \in S \qquad (10)$$

$$y_{ist} \in \{0, 1\} \qquad \forall i \in T, s, t \in S \qquad (11)$$

The first set of constraints ensures that each team meets each other team exactly once in a home game on one of the slots where the home team's venue is available. Consequently, each team will meet each other team exactly once in an away game as well, and these constraints are sufficient to construct a double round robin tournament. The next set of constraints make sure that each team plays at most once per time slot. Constraints (4)-(6) keep track of the number of days between two consecutive matches featuring the same team. The next set of constraints puts at least q calendar days between the two confrontations of a pair of teams. Constraints (8) enforce that a team plays at most two matches in a period of R_{max} slots. Constraints (9) make sure that there is no match between two teams on a particular time slot if the home team does not have its venue available, or if the away team marked this time slot in its forbidden game set. The final sets of constraints state that all variables are binary.

3 Results

We develop a heuristic algorithm, based on locally optimizing the schedule for each team. We show how this can be done efficiently by solving assignment problems. We make use of benchmark problem instances provided by Schönberger et al. [6] to compare our algorithm with IBM Ilog Cplex.

Finally, we apply our algorithm on a real-life case. The "LZV Cup" is an amateur indoor football league, which involves 75 teams, all situated in the vicinity of Leuven (Belgium). These teams are grouped according to their strength into 5 divisions, with 15 teams each. In each division, a double round robin tournament is played from September 1st to May 31st. In the opinion of the league organizers, it suffices to have 3 days between two successive matches for a team (i.e. $R_{max} = 3$), but there should be 60 days between two matches featuring the same teams (q=60). We compare our results with manually drafted schedules.

References

1. Bartsch, T., Drexl, A., and Kroger, S., Scheduling the professional soccer leagues of Austria and Germany, Computers and Operations Research, 33(7): 1907–1937, 2006
2. Della Croce, F. and Oliveri, D., Scheduling the Italian Football League: an ILP-based approach, Computers and Operations Research, 33(7): 1963–1974, 2006
3. Goossens, D. and Spieksma, F.C.R., Scheduling the Belgian soccer league, Interfaces, 39(2): 109–118, 2009
4. Goossens, D. and Spieksma, F.C.R., Soccer schedules in Europe: an overview, Journal of Scheduling, 15(5): 641–651, 2012
5. Knust, S., Scheduling non-professional table-tennis leagues, European Journal of Operational Research, 200: 358–367, 2010
6. Schönberger, J., Mattfeld, D.C. and Kopfer, H., Memetic algorithm timetabling for non-commercial sport leagues, European Journal of Operational Research, 153: 102–116, 2004

MISTA 2013

Baggage Scheduling at Airports

A framework for efficiently allocating resources to flight events

Torben C. Barth · Frauke Böckmann · Alexander Schäfer

1 Introduction

Baggage handling at airports is a critical sub process during the ground time of an aircraft. Especially, at large hub airports where many passengers change the aircraft the baggage handling processes are a critical bottleneck. Decisions on the baggage flow have a severe impact on baggage handling quality and in some cases even on flight operations. Today, we observe the need for a powerful decision support for the different baggage handling processes. The aim of this work is to combine different scheduling problems in such a way that a good overall solution can be provided.

Baggage handling consists of three main processes. One process is the handling of the bags of passengers ending their journey at the airport (called *inbound baggage handling* in the following). Furthermore, there is the process of handling the luggage of passengers transferring aircraft (*transfer baggage handling*). Finally, there is the process of handling the bags of departing passengers (*outbound baggage handling*). This process combines baggage flows from transfer and local check-in.

The processes have in common that the decision problems of assigning bags to resources are special types of scheduling problems. There is a similarity to the resource constrained project scheduling problem (RCPSP) since tasks have to be assigned to limited resources over time. In most cases the scheduling problems of baggage handling differ from the definitions of the known types of scheduling problems. Examples for differences are a multi-objective function and soft constraints. These differences are the reason that often specialized models are used in the context of baggage handling.

For decision problems at baggage handling it is very typical that the resources are not assigned exclusively to tasks (flights). Furthermore, the multi criteria objective combines the needs of the stakeholders and efficiency. Another important issue is the large degree of uncertainty at the airport.

Torben C. Barth
DTU Management Engineering, Produktionstorvet 424, DK-2800 Kgs. Lyngby, Denmark
E-mail: tocb@dtu.dk

Torben C. Barth, Frauke Böckmann and Alexander Schäfer
Fraport AG, D-60547 Frankfurt, Germany
E-mail: t.barth@fraport.de

2 Approach

The presented work consists of two steps. The first step is to introduce and describe the different sub problems and models for baggage scheduling. The second step is to define a framework for solving the different sub problems in an integrated way. We describe a framework leading to a global coordination of the identified scheduling problems.

The inbound, outbound and transfer baggage scheduling can be modeled as MIP models. The objective of each model consists of several direct cost terms denoted by c_i^{direct} and the violation of soft constraints y_r with the associated cost terms c_r^{soft}. For each flight f the set A_f of all feasible assignments to the available resources is generated. The model consists of a set partitioning constraint for each flight and several soft constraints connected to different resources r with the connected capacities CAP_r. A basic model is given below. Decision variable x_a indicates if assignment a is selected and u_{ar} the use of resource r by assignment a.

$$minimize \sum_{a,i} c_i^{direct} x_a + \sum_r c_r^{soft} y_r \tag{1}$$

$$\sum_{a \in A_f} x_a = 1 \quad \forall f \in F \tag{2}$$

$$x_a \in \{0,1\} \quad \forall a \in A \tag{3}$$

$$\sum_a u_{ar} x_a = CAP_r + y_r \quad \forall r \in R \tag{4}$$

$$y_r \geq 0 \quad \forall r \in R \tag{5}$$

3 Sub Processes

3.1 Inbound Baggage

For the inbound baggage the decision problem is to assign a baggage carousel to each arriving flight. A detailed description of the corresponding MIP model can be found in Barth and Böckmann [5] and Barth [6]. The direct cost of the problem are airline preferences, transportation time and suitability of the baggage carousels for a specific flight. As soft constraints we identify the avoidance of simultaneous handling, a balanced use of the resources and the capacity of the displays at the baggage carousels. Further approaches were presented by Delonge [8] and Frey et al. [10].

3.2 Outbound Baggage

For the outbound baggage it needs to be decided where and when the baggage should be handled. At the selected handling facilities the bags will be removed from the baggage handling system and loaded to containers for the transport to the aircraft. A feasible assignment is limited by the handling speed and capacity at the handling facilities. The process can strongly differ between airports. Most important the location –

decentral (at the parking position) or central (at the terminal building) – differentiates the problem structure. Direct cost in the objective are airline preferences and transportation time. Soft constraints of the problem are a balanced use of the resources, the capacity for placing containers at the handling facilities and number of flights which can be handled at one handling facility.

A detailed discussion of the problem and solution approaches can be found in Barth and Pisinger [4]. Approaches for different set-ups were published by Abdelghany et al.[1], Ascó et al. [2], and Frey et al. [9].

3.3 Transfer Baggage

For all containers with transfer bags of arriving flights an infeed station into the baggage handling system needs to be assigned. For choosing the best infeed station the parking position and the connecting flights are the most important information. Direct cost in the objective are the minimization of missed connections and of transportation time. The soft constraints concern the balancing of workload between the different infeed stations and the capacities. A solution approach basing on the MIP model was presented by Barth [3]. Kiermaier et al. [12] describe a solution for a different set-up.

3.4 Workforce and Task Scheduling

The workforce and task scheduling for the baggage handling staff at airports depends heavily on the decisions of the described processes. The baggage processes and the decisions on the baggage flow define place and time of staff demand. The literature in connection with ground staff scheduling is mainly concerned with the tactical problems of shift generation and shift assignment. Examples are the work of Herbers [11] and Clausen [7]. Clausen identified the assignment of the workload to the staff at the day of operations as a task scheduling problem [7]. The aim is to cover the defined workload as good as possible with the existing staff. Challenges for solving the problem are the dynamic changes and a special type of workload constraint. The workers cannot work for longer periods at their capacity limit. Furthermore, the task duration depends on the number of assigned workers. The staff can be assigned to all of the described processes respecting qualification profiles. The defined workload consists mostly on the combination of events like the handling of bags of different flights at the same time.

4 Baggage Scheduling Framework

In a global sense it is necessary to decide if the staff should move to the work or the work should be assigned to a location where staff is available. The most important connection between the processes is the assignment of the staff.

The goal of the framework is to fulfill all tasks with the best possible quality. Therefore a prioritization of the tasks is necessary. We use an iterative approach which takes the different planning steps during the operations into account. The baggage handling processes need to communicate their demand of staff. Then the resource allocation problem is solved and the results are communicated to the sub problems. The time horizon of the sub problems have major influence on the iterative process.

In special, it is not possible to move staff over long distances on short notice – but it could be still possible to assign the work to another location.

5 Results

For inbound, outbound and transfer baggage scheduling the described approaches are already implemented at Frankfurt Airport. The solution are heuristics based on exact solutions of the corresponding MIP models. Operational use of the workforce and task scheduling solution and the global framework is expected in 2014.

Our experiences show that the scheduling solutions for the sub problems improve the decisions. The objective functions can be improved about 10 % for each sub problem in comparison to manual decision making without optimization support. Furthermore, we observe a better balancing of the different criteria, since the human decision makers focus on the most important criteria.

When implementing the framework, we expect a improvement of the global objective by 10 % which was shown in first theoretical studies. The improvement could mean a reduction of staff or an increase in baggage handling quality.

References

1. Abdelghany A, Abdelghany K, Narasimhan R, Scheduling baggage-handling facilities in congested airports, Journal of Air Transport Management, 12(1), 76 - 81 (2006)
2. Ascó A, Atkin J, Burke E, The Airport Baggage Sorting Station Allocation Problem, In proceedings of the 5th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2011), 9-11 August 2011, Phoenix, Arizona, USA, 419 - 444 (2011)
3. Barth T, A model for the transfer baggage problem at airports, In Proceedings International Annual Conference of the German Operations Research Society in Hannover (2012)
4. Barth T, Pisinger D, Scheduling of outbound luggage handling at airport, Operations Research Proceedings 2011, 251 - 256 (2012)
5. Barth T, Böckmann F, Baggage carousel assignment at airports: Model and case study, In Proceedings International Conference for Airport Operations Management in Munich, 27 - 30 (2012)
6. Barth T, Optimal assignment of baggage carousels at airports, Technical Report, DTU Management Engineering, Report 4.2013 (2013)
7. Clausen T, Pisinger D, Airport Ground Staff Scheduling, PhD thesis, Technical University of Denmark (DTU), Kgs. Lyngby, Denmark. DTU Management 2011, no. 2 (2010)
8. Delonge F, Balancing load distribution on baggage belts at airports, In Proceedings International Annual Conference of the German Operations Research Society in Hannover (2012)
9. Frey M, Artigues C, Kolisch R, Lopez P, Scheduling and planning the outbound baggage process at international airports, In Proceedings IEEE International Conference on Industrial Engineering and Engineering Management (IEEM 2010) (2010)
10. Frey M, Kiermaier F, Kolisch R, Assignment of Local Baggage Streams to Claiming Carousels at Airports, In Proceedings International Conference for Airport Operations Management in Munich, 28 - 34 (2012)
11. Herbers J, Models and Algorithms for Ground Staff Scheduling On Airports, PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, Faculty of Mathematics, Computer Science and Natural Sciences (2005)
12. Kiermaier F, Kolisch R, Planning the transfer baggage handling at airports, In Proceedings International Conference for Airport Operations Management in Munich, 23 - 26 (2012)

MISTA 2013

Solving a time-indexed formulation by preprocessing and cutting planes

L. Berghman · F.C.R. Spieksma · V. T'Kindt

1 Introduction

The time-indexed formulation for single machine scheduling problems is well studied in the literature. Seminal work of [3] and [5], and further work by [2] and [7] have resulted in a large body of polyhedral results for the time-indexed formulation. As far as we are aware, all this polyhedral knowledge has not been applied to time-indexed formulations of scheduling problems with multiple machines.

We consider the problem of scheduling n jobs on m unrelated parallel machines within a given timespan. Each machine can handle at most one job at a time and preemption is not allowed. Each job i is defined by a processing time p_{ik} when scheduled on machine k, and a processing cost, denoted by c_{ikt} which also depends on the time period t at which it is started. We are interested in finding a feasible schedule minimizing the total processing cost. Notice that this enables to model many objective functions such as (weighted) sum of completion times, makespan, ... Consequently, the tackled problem is $\mathcal{NP}$-hard. Building a time-indexed formulation for that problem also enables to easily incorporate features such as release times and precedence relations.

We model this unrelated parallel machine scheduling problem as a single machine problem in the following way: by copying each job m times, we obtain nm tasks j. We define J as the set containing all tasks. This set can be partitioned in two different ways. First of all, we consider the subsets $J_i \subseteq J$ with $i = 1, \ldots, n$ containing all tasks related to job i. Secondly, we consider the subsets $J^k \subseteq J$ with $k = 1, \ldots, m$ containing all tasks related to machine k. Every subset $J_i \cap J^k$ consists of a single task j. The processing time of task $j = J_i \cap J^k$ equals $p_j = p_{ik}$. We denote by $c_{jt} = c_{ikt}$ the cost

L. Berghman
Université de Toulouse - Toulouse Business School, 20 BD Lascrosses BP 7010, 31068 Toulouse Cedex 7, France
E-mail: l.berghman@tbs-education.fr

F.C.R. Spieksma
ORSTAT, KULeuven, Naamsestraat 69, 3000 Leuven, Belgium
E-mail: frits.spieksma@econ.kuleuven.be

V. T'Kindt
Université François-Rabelais Tours, Laboratoire d'Informatique (EA 6300 et ERL CNRS 6305), 64 avenue Jean Portalis, 37200 Tours, France
E-mail: tkindt@univ-tours.fr

of starting task $j = J_i \cap J^k$ in time period t. Notice that specifying the task (index j), implies specifying the job and the machine, and vice versa.

For each task j and for each time period $t = 1, \ldots, T$, we define the decision variables

$$x_{jt} = \begin{cases} 1 & \text{if task } j \text{ starts in time period } [t; t+1[, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

An IP-model for this machine scheduling problem is the following:

$$\min \sum_{j=1}^{nm} \sum_{t=1}^{T} c_{jt} x_{jt} \tag{2}$$

subject to

$$\sum_{j \in J_i} \sum_{t=1}^{T} x_{jt} = 1 \qquad \forall i = 1, \ldots, n, \tag{3}$$

$$\sum_{j \in J^k} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \qquad \forall k = 1, \ldots, m; \forall t = 1, \ldots, T, \tag{4}$$

$$x_{jt} \in \{0, 1\} \qquad \forall j = 1, \ldots, nm; \forall t = 1, \ldots, T. \tag{5}$$

The objective function (2) minimizes the total cost. Constraints (3) state that out of the tasks related to job i, exactly one task has to be scheduled. The capacity constraints (4) state that: for each time period t, only one task out of the tasks related to machine k can be executed. We always only consider the time instances $1 \leq s \leq T$.

2 Valid inequalities

[5] and [8] present valid inequalities for the single machine problem denoted by P_1. Noteworthy, an inequality for P_1 can be extended to an inequality for the parallel machine problem, denoted by P_m ($m > 1$), by setting all coefficients that correspond to variables that involve tasks not related to some specific machine k ($1 \leq k \leq m$) to 0. Then, it is not difficult to observe that in this way, any inequality valid for P_1 can be extended to an inequality valid for P_m.

For introducing a new set of valid inequalities, we need the following notation. For each $j = 1, \ldots, nm$, we define $T(j)$ as the set of tasks that are related to the same machine as task j. Notice that $T(j)$ does not include task j. Moreover, we define $p_j^* = \max_{l \in T(j)} p_l$; thus p_j^* is the largest processing time of the tasks in $T(j)$.

For each pair of jobs $\{i, i'\} \in \{1, \ldots, n\}$, and for each pair of machines $\{k, k'\} \in \{1, \ldots, m\}$, let $j = J^k \cap J_i$, $q = J^k \cap J_{i'}$, $a = J^{k'} \cap J_i$ and $b = J^{k'} \cap J_{i'}$. For each quadruple of such four tasks, for all time periods $t_1, t_2 = 1, \ldots, T$, for all $\Delta_1 \in \{1, \ldots, \min\{p_j^*, p_q^*\}\}$ and for all $\Delta_2 \in \{1, \ldots, p_a^*\}$, we have the following inequalities:

$$\sum_{s=t_1-p_j+1}^{t_1+\Delta_1-1} x_{js} + \sum_{s=t_1-p_q+1}^{t_1+\Delta_1-1} x_{qs} + \sum_{l \in T(j) \setminus \{q\}} \sum_{s=t_1-p_l+\Delta_1}^{t_1} x_{ls} + \sum_{s=t_2-p_a+1}^{t_2+\Delta_2-1} x_{as} + \sum_{s=t_2-p_b+\Delta_2}^{t_2} x_{bs} \leq 2 \tag{6}$$

3 Preprocessing the time-indexed formulation

Hence, we apply reduced cost fixing to the time-indexed formulation (see [1] and [6] for other examples). This technique relies on the relation between an IP formulation and its linear relaxation LP. Let z^*_{LP} (resp. z^*_{MIP}) be the optimal LP solution value (resp. MIP) and let B^* be the associated basis. The preprocessing consists of deducing the value of boolean variables x_{jt}. We distinguish between the basic variables and the non-basic variables.

It is well known that, applied to our model, we have:

$$z^*_{MIP} = z^*_{LP} + \sum_{x_{jt} \notin B^*} r_{jt} x_{jt}, \tag{7}$$

with r_{jt} the reduced cost associated to variable x_{jt}. Let UB be an upper bound to z^*_{MIP}. Then, we can write:

$$\sum_{x_{jt} \notin B^*} r_{jt} x_{jt} \leq UB - z^*_{LP}. \tag{8}$$

From inequality (8) we can deduce the following fixing rule: $\forall x_{jt} \notin B^*$, if $r_{jt} > UB - z^*_{LP}$ then in any optimal solution of MIP, $x_{jt} = 0$. By reasoning on the slack variables associated to the x_{jt}'s we can deduce when $x_{jt} = 1$.

Concerning basic variables $x_{jt} \in B^*$, we use pseudo-costs l_{jt} and u_{jt} computed, for instance, by means of Driebeek's penalties (see [4]). These penalties correspond to lower estimates on the increase of z^*_{LP} if x_{jt} is set to 0 or 1. We have:

1. $\forall x_{jt} \in B^*$, if $(l_{jt} x_{jt}) \geq (UB - z^*_{LP})$ then $x_{jt} = 1$,
2. $\forall x_{jt} \in B^*$, if $(u_{jt}(1 - x_{jt})) \geq (UB - z^*_{LP})$ then $x_{jt} = 0$.

The preprocessing algorithm roughly works as follows: (1) Compute an upper bound, (2) solve the LP relaxation with added valid inequalities, (3) fix some variables x_{jt}. This process iterates until no more variable can be fixed. Then, the IP model is solved with the set of fixed variables included. We will present computational results of this algorithm, and report the size of the instances we can solve to optimality.

References

1. Baptiste P., Della Croce F., Grosso A., T'kindt V., "Sequencing a single machine with due dates and deadlines: An ILP-based approach", *Journal of Scheduling*, 13(1), 39–47 (2010)
2. Y. Crama and F. C. R. Spieksma, "Scheduling jobs of equal length: Complexity, facets and computational results", *Mathematical Programming*, 72, 207–227 (1996)
3. M. E. Dyer and L. A. Wolsey, "Formulating the single machine sequencing problem with release dates as a mixed integer problem", *Discrete Applied Mathematics*, 26, 255–270 (1990)
4. J. T. Linderoth and M. W. P. Savelsbergh, "A computational study of search strategies for mixed integer programming", *INFORMS Journal on Computing*, 11 (2), 173–187 (1999)
5. J. P. Sousa and L. A. Wolsey, "A time indexed formulation of non-preemptive single machine scheduling problems", *Mathematical Programming*, 54, 353–367 (1992)
6. T'kindt V., Della Croce F., Bouquard J.-L., "Enumeration of Pareto optima for a flowshop scheduling problem with two criteria", *Informs Journal on Computing*, 19(1), 64–72 (2007)
7. J. M. van den Akker and J. A. Hoogeveen and S. L. van de Velde, "Parallel machine scheduling by column generation", *Operations Research*, 47 (6), 862–872 (1999)
8. J. M. van den Akker and C. P. M. Van Hoesel and M. W. P. Savelsbergh, "A polyhedral approach to single-machine scheduling problems", *Mathematical Programming*, 85, 541–572 (1999)

MISTA 2013

Compilation of Maintenance Schedules based on their Key Performance Indicators for Fast Iterative Interaction

Dara Curran · Roman van der Krogt · James Little · Nic Wilson

1 Introduction

Generating a good schedule is a complex task, for which highly optimised methods and software have been developed, see, e.g., M.L.Pinedo (2008). However, when building a scheduling support tool, the clients' requirements and priorities are often unclear, even to themselves, so it takes significant effort to elicit these from the end-user (van der Krogt et al, 2009). Even worse, the priorities are often subjective and change between actual instances of a problem. Prompting the user with a schedule can generate feedback regarding what they want, and thus often an iterative approach is required. However, generating another schedule based on such feedback could take several minutes or even longer, which prohibits an extensive exploration of the available possibilities. It is therefore desirable to have ways of allowing the user to explore the possible solution space interactively—in particular, by putting bounds on the key performance indicators (KPIs)—before settling on a workable solution to move forward with. In order to facilitate this idea we need to generate sufficient compact compiled schedules in advance and to store them in such a way that makes navigation through them easy.

2 Compiling and Storing Schedules

Our system creates schedules by iteratively altering the duration of each task and finding valid schedules for each task duration combination (if one exists). The schedule generator begins by calculating the minimum and maximum duration that is possible for each task, given the resources available. It then breaks up the minimum/maximum range into a series of segments, resulting in a list of possible durations for each task. For each of these durations, the generator first checks to ensure that a schedule is possible for a given task duration. If it is not, the task duration in question is blacklisted and not employed further.

The generator then iterates through all possible combinations of task durations and attempts to generate schedules for each. For each combination of task durations, the generator may find a large number of solutions. Many of these solutions will represent schedules that

Dara Curran, Roman van der Krogt, James Little and Nic Wilson
Cork Constraint Computation Centre, University College Cork, Cork, Ireland
E-mail: n.wilson@4c.ucc.ie

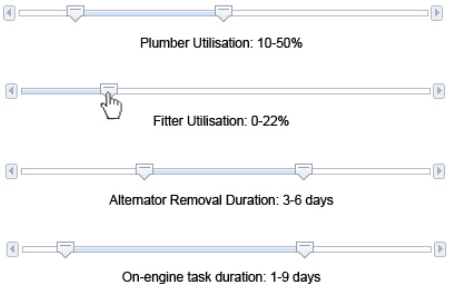


Fig. 1: Part of example User Interface

are very similar to one another which is not desirable from a user perspective. To limit this effect, each generated schedule is checked against previous schedules to ensure that it is significantly different.

If the schedule is considered sufficiently different, the information is stored using a Multi-valued Decision Diagram (MDD). MDDs generalise Binary Decision diagrams (BDDs, (Bryant, 1995)) to non-boolean values, and have been studied for example by Wilson (2005). An MDD is a directed graph with a unique *source* (i.e., initial) node and a unique *sink* (i.e., final) node. Each edge is associated with an assignment to a variable. Paths from Source to Sink correspond with assignments to a set of variables, and so the MDD represents a set of complete assignments via its set of paths. This can be a very compact representation, since the number of complete assignments represented can even be exponential in the number of nodes in the MDD.

The MDD is created from the schedules as follows. We have one variable for each KPI. The values of the variable represent small ranges of the KPI; for example, for the KPI "Utilisation of Plumbers" we might use a variable with five values, with the first value representing 0-20% utilisation. A path in the MDD then corresponds with an assignment to all the KPI variables, and this combination must be feasible, i.e., we have found a schedule which has that combination of values of the KPIs.

3 Usage

Once the MDD is created, the system can efficiently retrieve from it the possible ranges of the KPIs, and show what is available in terms of possible schedules and their performance indicators. This is done through sliders, as depicted in Figure 1. Initially, the sliders represent the ranges for which solutions exist. However, the user can change them to reflect their desires. The MDD can then be used to update the ranges of the indicators by enforcing the constraints proposed by the user. In parallel, each solution in the form of a set of KPIs has an associated actual schedule which the user can inspect at any time. This gives the scheduler an end view to make a further judgement of choosing this plan or continuing searching.

Once the user has set KPIs to their satisfaction and obtained a set of schedules from the system, they may find that while some schedules fit their constraints, they are prepared to relax certain KPI values in order to achieve better results elsewhere. This is similar to solution critiquing in recommender systems (McGinty and Reilly, 2011). However, adjusting

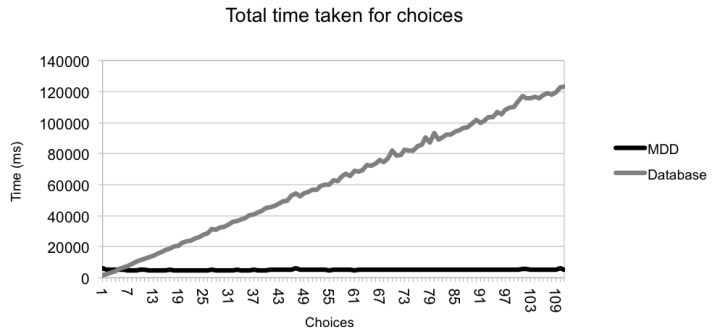


Fig. 2: Total time take for choices

KPI values manually at this point would become a trial-and-error process and so we devised a simple local search mechanism to perform this type of search automatically. Once the user has selected a schedule of interest, they have the option of selecting a KPI that they would like to either increase or decrease in value. The system then iteratively increases each KPI value range until the desired KPI has shifted in value, at which point the system presents a new set of schedules.

4 Initial Results and Discussion

The approach presented here is included in a system to support the Irish Navy in solving a maintenance scheduling problem. Their maintenance/refit policy across all their ships is based on a 28 day period (or 20 working days) every year. During that time, a team of specialist fitters, riggers, electricians and plumbers are employed at the dockyard to carry out the majority of tasks associated with the maintenance. Ideally, the schedule will have all the tasks completed within the time window without the need for unplanned outsourcing. Regarding the evaluation of a schedule, there are many criteria of interest, expressed as KPIs, representing utilisation of different resources, durations of activities, and so on. The scheduling problem is described in detail in Boyle et al (2011).

The justification for building an MDD to store possible user choices about schedules is motivated primarily by the efficiency at which such choices can be made and the model updated. In order to test whether our MDD approach has been successful in this regard, we conducted experiments to compare the approach with the most obvious alternative of employing a database directly to search for solutions each time a user makes an adjustment to a KPI value. For example, Figure 2 shows how the MDD approach compares to a pure database system using SQL queries to retrieve the new minumum and maximum value ranges for each KPI as each choice is made, for varying number of choices. This results were obtained using actual dockyard data and consist of 64 tasks taking place over a fixed makespan utilising 47 resources. For our experiment we consider 2 types of KPI: task duration and resource utilisation (i.e. the amount of time that a resource is utilised over the entire makespan). This gives a total of 111 KPIs and the experiments employ 1329 schedules. (In practice, a human scheduler may choose to focus on a smaller set of KPIs.) The results show that although the MDD approach brings some overhead, for anything but very few choices

the MDD approach is most efficient. As the number of choices increases, the time take for the database approach to find new minimum and maximum ranges for each KPI increases linearly, while the MDD approach remains almost static.

These and other results show that the approach can work well for this type of problem. Future work includes a broader exploration of our methodology. Having shown promising results in one domain, we are keen to explore other types of scheduling problems to see if we can achieve the same results. We are also looking into cleverer ways of finding the initial set of solutions. This is a time-consuming step, so any reduction in the number of schedules we need to compute, while retaining the same or similar coverage would be a big benefit.

References

Boyle G, Little J, Manning J, van der Krogt R (2011) A constraint-based approach to ship maintenance for the irish navy. In: Proceedings of the Irish Transport Research Network Conference (ITRN-11)

Bryant RE (1995) Binary Decision Diagrams and beyond: enabling technologies for formal verification. In: Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design, pp 236–243

van der Krogt R, Little J, Simonis H (2009) Scheduling in the real world: Lessons learnt. In: Proceedings of the ICAPS'09 Scheduling and Planning Applications workshop

McGinty L, Reilly J (2011) On the evolution of critiquing recommenders. In: Ricci F, Rokach L, Shapira B, Kantor PB (eds) Recommender Systems Handbook, Springer, pp 419–453

MLPinedo (2008) Scheduling: Theory, Algorithms and Systems (3rd edition). Springer

Wilson N (2005) Decision diagrams for the computation of semiring valuations. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), pp 331– 336

MISTA 2013

Reentrance and exact time lag on flow shop scheduling problem

Amrouche Karim · Boudhar Mourad

1 Introduction

We propose a study of two machines flow shop with reentrance and exact time lag L denoted by $F2|chain - reentrant, l_j = L|C_{max}$. Each task may be processed in this order M_1, M_2, M_1 and there is an exact time lag L between the completion time of the first operation and the start time of the second operation on the first machine (called also primary machine), the objective is to minimize the makespan. Note that the problem $F2|chain - reentrant|C_{max}$ is equivalent to $V2||C_{max}$ [1]. For the two-machine flow shop without reentrance, the makespan criterion is the most studied in the literature. The simplest case occurs when all jobs have the same time lag L i.e. the task j cannot be started on the second machine before $l_j = L$ units of time of its completion time on the first machine. We can easily show that this problem is equivalent to the case without time lags by shifting all tasks to the second machine with L units of time, therefore the Johnson's algorithm [2] gives the optimal solution. Consider now the case where all time lags are different. The problem $F2|l_j|C_{max}$ was studied by Mitten [3]. He proved that if we restrict to permutation schedules, the optimal solution is given by Johnson's algorithm with modifications of processing times. But in the general case (without restriction to permutation schedules), minimization of makespan is NP-Hard, Dell'Amico [4] Yu et al. [5] Lawler et al. [6] have proved that this problem is NP-Hard even if the processing time of jobs are units. Yu [7] proposed a condition for which the permutation scheduling becomes dominant, i.e.: $\forall i, j\ l_i \leq l_j + max\{p_{1j}, p_{2j}\}$. Other researchers are interested in coupled tasks scheduling problems denoted (CT), a coupled task is composed by two distinct operations executed in order, separated by a time lag. In [8], Shapiro discussed practical cases of (CT) and gave heuristics with

Amrouche Karim
Affiliation
RECITS laboratory, USTHB University, Faculty of Mathematics
BP 32 Bab-Ezzouar, El-Alia 16111, Algiers, Algeria
E-mail: amrouche-karim@hotmail.com

Boudhar Mourad
Affiliation
USTHB University, Faculty of Mathematics
BP 32 Bab-Ezzouar, El-Alia 16111, Algiers, Algeria
E-mail: mboudhar@yahoo.fr

numerical experiments. Orman and Potts studied the same problem in one machine [9], whose objective is minimizing makespan. They identified several problems and hierarchies according to their complexity. Ahr et al. [10] proposed an exact algorithm using dynamic programming to solve the problem of (CT) in a machine for small instances when L is fixed; this problem is denoted $1/coupled-task, a_i = a, l_i = L, b_i = b/C_{max}$. Ageev and Baburin [11] proposed a $7/4$ and $3/2$-approximation to solve problem $1/coupled-task, a_i = b_i = 1, l_i = L/C_{max}$ and $F2/a_i = b_i = 1, l_i = L/C_{max}$. Few works were performed by adding constraints to coupled tasks. Blazewicz et al. [12] showed that the polynomial problem $1/coupled-task, a_i = b_i = 1, l_i/C_{max}$ is NP-Hard by adding a precedence constraint between coupled tasks.

In this paper we prove the NP-hardness of a special case and we give some special subproblems that can be solved in polynomial time. Also, we propose four heuristics with numerical experiments. For scheduling tasks denote:

- a_i: the processing time of the first operation of the task T_i on the first machine.
- b_i: the processing time of the task T_i on the second machine, $b_i \leq L$.
- c_i: the processing time of the second operation of the task T_i on the first machine.

2 NP-hardness

We focus our study on the complexity of the problem in case $a_j = c_j = const$; the problem is NP-hard in the strong sense by a reduction from the 3-partition problem.

Theorem 1 *The problem $F2/chain - reentrant, a_j = c_j = c, l_j = L/C_{max}$ is NP-Hard in the strong sense.*

3 Polynomial subproblems

We show here that when processing times are constants or greater than $L/2$, the problem is solved by a polynomial algorithm based on the maximum weight matching.

Theorem 2 *The problem $F2/chain - reentrant, a_j > L/2, c_j > L/2, l_j = L/C_{max}$ reduces to the maximum weight matching.*

Theorem 3 *The problem $F2/chain - reentrant, b_j = l_j = L/C_{max}$ reduces to the maximum weight matching.*

Proposition 1 *The problem $F2/chain - reentrant, a_j = b_j = c_j = l_j = L/C_{max}$ is polynomially solved by choosing any order of tasks.*

4 Heuristics and numerical experiments

We propose four heuristics for the general problem based on the list scheduling approach. We start by giving some lower bounds for the optimal makespan of this problem. Let T be the set of partitioned tasks on two subsets ($T = T_1 \cup T_2$) such that $T_1 = \{T_i / \min\{a_i, c_i\} > L\}, |T_1| = n_1$, $T_2 = T \setminus T_1$ and $|T_2| = n_2 = n - n_1$.

Proposition 2 $LB_1 = \min\limits_{1\leq i\leq n}\{a_i\} + \sum\limits_{i=1}^{n_2} b_i + \min\limits_{1\leq i\leq n}\{c_i\} + n_1 L + \sum\limits_{i=1}^{n1}(a_i + c_i)$ *is a lower bound.*

Proposition 3 $LB_2 = n_1 L + \sum_{i=1}^{n} (a_i + c_i)$ *is a lower bound.*

Proposition 4 $LB_3 = \alpha \left(\min_{1 \le i \le n} \{a_i\} + L + \left(\left[\frac{L}{\min_{1 \le i \le n} \{a_i, c_i\}} \right] + 1 \right) \min_{1 \le i \le n} \{c_i\} \right)$ $+ \min_{1 \le i \le n} \{a_i\} + L + \beta \min_{1 \le i \le n} \{c_i\}$ *is a lower bound.*

where $\alpha = \left[n / \left(\left[\frac{L}{\min_{1 \le i \le n} \{a_i, c_i\}} \right] + 1 \right) \right]$ is the number of batches that contain $\left[\frac{L}{\min_{1 \le i \le n} \{a_i, c_i\}} \right] + 1$ interlaced tasks, and $\beta = n - \left(\left[\frac{L}{\min_{1 \le i \le n} \{a_i, c_i\}} \right] + 1 \right) \alpha$ is the number of remaining tasks that are interlaced together in the last batch.

Consequently, $LB = max(LB_1, LB_2, LB_3)$ is a lower bound.

The principle of the four heuristics is as follows:

Since the set of tasks is not empty, form a new batch:

- choose the task that has the smallest c_i (or a_i) and interlace at right (or at left) the maximum number of tasks.
- Calculate the completion time of each formed batch j denoted $C_{max_B atch j}$.
- The completion time of all tasks is equal to the sum of $C_{max_B atch j}$.

In order to assess the performance of the different heuristics, several instances are randomly generated according to the uniform law and by using different combinations of tasks. After the completion of several tests, we found that globally heuristic H_1 is better than the others.

References

1. Lev V and Adiri I. V-shop scheduling. European J. Opnl. Res. 1984; 18:51–56.
2. Johnson SM. Optimal two and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1954; 1:61–68.
3. Mitten LG. Sequencing n jobs on two machines with arbitrary time lags. Management Science 1959; 5:293–298.
4. Dell'Amico M. Shop problems with two machines and time lags. Operations Research 1996; 44(5):777–787.
5. Yu W, Hoogeveen H, and Lenstra JK. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is np-hard. Journal of Scheduling 2004; 7:333–348.
6. Lawler EL, Lenstra JK, Rinnooy Kan AHG, and Shmoys DB. Sequencing and scheduling theory : algorithms and complexity. In S.C. Graves, P.H. Zipkin, and A.H.G Rinnooy Kan editors, Handbooks in Operations Research and Management Science, North Holland: Amesterdam, 1993.
7. Yu W, The two-machine flowshop problem with delays and the one-machine total tardiness problem. PhD thesis, Technische Universiteit Eindhoven, 1996.
8. Shapiro RD. Scheduling coupled tasks. Naval Research Logistics Quarterly 1980; 27(2):489–97.
9. Orman AJ, Potts CN. On the Complexity of Coupled-task Scheduling.Discrete Applied Mathematics 1997; 72:141–54.
10. Ahr D, Békési J, Galambos G, Oswald M, Reinelt G. An exact algorithm for scheduling identical coupled tasks. Mathematical Methods of Operations Research (ZOR) 2004; 59:193–203.
11. Ageev AA, Barburin AE. Approximation algorithms for UET scheduling problems with exact delays, Operations Research Letters 2007; 35:533-540.
12. Blazewicz J, Ecker K, Kis T, Potts CN. Tanas M and Whitehead J, Scheduling of coupled tasks with unit processing times, Journal of Schuduling 2010; 13:453-461.

MISTA 2013

A GRASP approach for the Job Shop Scheduling Problem with minimizing the Total Weighted Tardiness

Jens Kuhpfahl · Christian Bierwirth

1 Introduction

Due to the increased importance of meeting delivery dates in supply chains, scheduling problems have been extended to the consideration of due dates in the recent literature. Missing job due dates can cause considerable penalty cost in practice. With this motivation, the paper addresses the job shop scheduling problem with due dates and the minimization of the total weighted tardiness as objective.

In the job shop scheduling problem (JSP), a set of jobs has to be processed on a set of machines. Each job is characterized by a given sequence of operations, each assigned to a specific machine for a given duration. Every machine can perform at most one operation at a time. Preemption in the processing on a machine is not allowed. Next to ready times, that have to be respected, a due date d_j is attached to each job. The total weighted tardiness is defined by: TWT $= \sum_{j=1}^{n} w_j \cdot \max\{0, C_j - d_j\}$ where w_j is the weight of job j and C_j is its completion time in the schedule.

In the literature, only a few papers treat the JSP with TWT objective (herein after referred to as JSPTWT). Besides an exact method [16], current research mainly focusses on metaheuristic concepts, see e.g. [4, 6, 10, 12, 15]. In this work, we present a powerful solution procedure for solving the JSPTWT. The algorithmic design of the procedure is geared to the structure of the classic greedy randomized adaptive search procedure (GRASP). In Section 2 we describe the design of the main components of our GRASP approach. The performance of the proposed heuristic is studied in Section 3. Finally, we end up with a conclusion.

2 The GRASP design

The heuristic concept of GRASP and algorithmic variants [7] have been applied successfully to a plenty of combinatorial optimization problems [8], including the standard minimum makespan JSP [1, 3]. In general, an iterative search process forms the basic architecture of GRASP. In every iteration, a feasible schedule is built in a construction

Jens Kuhpfahl
School of Economics and Business, Martin-Luther-University Halle-Wittenberg, Germany
E-mail: jens.kuhpfahl@wiwi.uni-halle.de

Christian Bierwirth
School of Economics and Business, Martin-Luther-University Halle-Wittenberg, Germany
E-mail: christian.bierwirth@wiwi.uni-halle.de

phase. Afterwards, in the improvement phase, the initial solution is transferred into a local optimum by means of a local search. The best solution found in the iterative process is returned as the final result. Additionally, our GRASP heuristic runs in two phases: in early iterations, the so called first part (8000 iterations), we determine the best setting of GRASP to the individual test problem. In later iterations, the so called second part, we restrict the search on the found setting.

In the construction phase, our GRASP implementation applies the hybrid scheduling algorithm A1 of Storer et al. [17]. An active schedule is generated by consecutively inserting so far unscheduled operations in a partial solution. In this way, the number of schedulable operations is limited, forming a so called restricted candidate list. After that, a priority rule is used to order the list. The particular operation that is scheduled next is drawn by a stochastic sampling, evaluating the position of operations in the list. In the first part of a GRASP run, each of the eight priority rules ATC, SPT, WSPT, EDD, ODD, SPT+SRPT, PT+PW, PT+PW+ODD [9] is used for the construction of 1000 initial solutions. In the second part, only the best performing priority rules are used in the construction phase of the GRASP iterations. In this way, already the construction phase takes up an adaptive control mechanism.

In the improvement phase, GRASP performs a local search procedure with a First-Improve scheme. The local search procedure is based on a disjunctive graph representation for the JSPTWT. This graph model adapts the well-known disjunctive graph representation for the standard JSP towards the needs of the TWT objective [11]. In the modified disjunctive graph representation, a solution comprises to a couple of longest paths which together form a critical tree. The critical tree allows an efficient identification of the critical blocks [14] nesting inside the solution. After the critical blocks of a solution have been identified, we determine the frequency of a block belonging to different longest paths. In the subsequent neighbourhood search, those critical blocks are analyzed at first which show the highest frequency. In this way, a multiple of longest paths is potentially reduced by the first improving neighbourhood move.

In the local search procedure, two neighbourhood operators are applied alternately. As one neighbourhood, we use the critical shift operator, introduced by [5]. This neighbourhood operator shifts an operation of a critical block to the most front or most end of this block with regard to the feasibility of the schedule. As the second neighbourhood we use the extended critical shift operator, introduced by [11], which is an extension of the multi-swap operator [13]. Here, a neighbour is generated either by a swap at the start or the end of a critical block, or by a shift of an operation to the most end of the critical block. Furthermore, additional swaps on the machines of the preceding and succeeding operations are executed if there is no idle time between the two corresponding operations on these machines. The local search procedure ends up in a local optimum with respect to both neighbourhood definitions.

The GRASP heuristic takes advantage from two embedded adaptive components. As described above, priority rules are adaptively chosen for the construction of schedules in the second part of the GRASP run. As a further adaptive component, we derive a threshold value from the best objective function value observed in the first part of GRASP. If the improvement phase ends up with a solution quality better than the threshold, the related initial solution is used further on in a couple of subsequent iterations. This mechanism enables intensifying the search in highly promising regions of the search space. Within such periods we suspend the above described critical block control feature and use a random access for the selection of critical blocks in the local search. In this way, the surrounding of a local optimum is explored intensively.

	$f = 1.3$				$f = 1.5$				$f = 1.6$			
Instance	best	GRASP	GLS	ILS	best	GRASP	GLS	ILS	best	GRASP	GLS	ILS
ABZ05	1403	*	*	*	69	*	*	*	0	*	*	*
ABZ06	436	*	*	*	0	*	*	*	0	*	*	*
LA16	1169	*	*	*	166	*	*	*	0	*	*	*
LA17	899	*	*	*	260	*	*	*	65	*	*	*
LA18	929	*	*	*	34	*	*	*	0	*	*	*
LA19	948	*	*	*	21	*	*	*	0	*	*	*
LA20	805	*	*	*	0	*	*	*	0	*	*	*
LA21	463	*	*	*	0	*	*	*	0	*	*	*
LA22	1064	*	*	*	196	*	*	*	0	*	*	*
LA23	835	*	*	*	2	*	*	*	0	*	*	*
LA24	835	*	*	*	82	*	*	*	0	*	*	*
MT10	1363	*	*	*	394	*	*	*	141	*	*	*
ORB01	2568	*	2570	*	1098	*	*	1196	566	*	604	604
ORB02	1408	*	*	*	292	*	*	*	44	*	*	*
ORB03	2111	2131	*	*	918	*	*	952	422	*	*	*
ORB04	1623	*	*	*	358	*	*	*	66	*	*	*
ORB05	1593	1667	*	1738	405	*	*	*	163	*	181	*
ORB06	1790	*	*	*	426	*	*	*	28	*	*	*
ORB07	590	608	*	*	50	*	*	*	0	*	*	*
ORB08	2429	2453	2439	2461	1023	1085	*	*	621	657	646	*
ORB09	1316	*	*	*	297	*	*	*	66	*	*	*
ORB10	1679	*	*	*	346	*	*	424	76	*	84	78

Table 1 Experimental results: best known values and results of GRASP, GLS [6] and ILS [12]

3 Computational Results

In order to evaluate our approach, we apply GRASP to 18 benchmark instances that are provided by the OR-Library [2]. This benchmark set is often used for the comparison of solution methods for the JSPTWT (see e.g. [10, 15]). The test instances consist of 10 jobs and 10 machines. Weights are distributed related to a specific key, due dates are calculated with the help of a due date factor f, cf. [10].

We compare the results of GRASP with those of the Genetic Local Search Algorithm (GLS) from Essafi et al. [6] and the Iterated Local Search Procedure (ILS) from Mati et al. [12] as these are the best known metaheuristics for JSPTWT. They report results for 10 runs of GLS as well as 10 runs of ILS, each consuming 18 seconds runtime (PC with 2.8 GHz/2.6 GHz). To ensure a fair comparison, our GRASP, which takes just a single run, is given a time limit of 138 seconds (PC with 3.4 GHz).

The achieved results are presented in Table 1. For three due date factors $f = 1.3, 1.5, 1.6$, the best known values are taken from the literature ("best") and are compared with the outcome of GRASP, GLS and ILS. An asterisk (*) indicates that a solution has been found which corresponds to the best known objective value.

It can be taken from the table that GRASP comes along with the best known solutions in 60 of 66 cases. The total gap of the GRASP solutions (= sum of the TWT deviations between the heuristic and the best known solution) results in 234. At the same time, GLS finds the same number of best known solutions but shows a better total gap of 101, while ILS is able to identify only 59 best solutions and produces a clearly worse total gap of 427. To summarize, GRASP is significantly superior to ILS but is dominated by GLS with respect to the total gap.

4 Conclusion and Outlook

In our talk, we describe the GRASP implementation for JSPTWT in detail. We will discuss the algorithmic design and analyze its impact on the achievable quality of solutions. The computational results observed so far constitutes that GRASP describes a promising solution approach for JSPTWT which justifies further research in this field.

References

1. R. M. Aiex, S. Binato, and M. G. C. Resende. Parallel grasp with path-relinking for job shop scheduling. *Parallel Computing*, 29:393–430, 2003.
2. J. E. Beasley. Or-library. http://people.brunel.ac.uk/ mastjjb/jeb/info.html, 2013.
3. S. Binato, W. J. Hery, D. Loewenstern, and M. G. C. Resende. A greedy randomized adaptive search procedure for job shop scheduling. In C. C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 58–79. Kluwer Academic Publishers, 2002.
4. K. Bülbül. A hybrid shifting bottleneck-tabu search heuristic for the job shop total weighted tardiness problem. *Computers & Operations Research*, 38:967–983, 2011.
5. M. Dell'Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252, 1993.
6. I. Essafi, Y. Mati, and S. Dauzère-Pérès. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research*, 35:2599–2616, 2008.
7. P. Festa and M. G. C. Resende. An annotated bibliography of grasp - part i: Algorithms. *International Transactions in Operational Research*, 16:1–24, 2009.
8. P. Festa and M. G. C. Resende. An annotated bibliography of grasp - part i: Applications. *International Transactions in Operational Research*, 16:131–172, 2009.
9. M. S. Jayamohan and C. Rajendran. New dispatching rules for job shop scheduling: a step forward. *International Journal of Production Research*, 38:563–586, 2000.
10. S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3:125–138, 2000.
11. J. Kuhpfahl and C. Bierwirth. A new neighbourhood operator for the job shop scheduling problem with total weighted tardiness objective. In *Proceedings of International Conference on Applied Mathematical Optimization and Modelling*, pages 204–209, 2012.
12. Y. Mati, S. Dauzère-Pérès, and C. Lahlou. A general approach for optimizing regular criteria in the job-shop scheduling problem. *European Journal of Operational Research*, 212:33–42, 2011.
13. H. Matsuo, C. J. Suh, and R. S. Sullivan. A controlled search simulated annealing method for the general jobshop scheduling problem. *Working paper 03-04-88. University Texas*, 1988.
14. E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813, 1996.
15. M. Pinedo and M. Singer. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 46:1–17, 1999.
16. M. Singer and M. Pinedo. Computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, 29:109–119, 1998.
17. R. H. Storer, S. D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38:1495–1509, 1992.

MISTA 2013

Time-Hierarchical Scheduling

A worst-case analysis of a rolling hierarchical approach for integrated production planning and scheduling in make-to-order environments

Jens Leoff · Heiner Ackermann · Karl-Heinz Küfer

Introduction

Production planning in a make-to-order environment is a complex task to be executed regularly. Complexity accrues from many different aspects, amongst others from the following two: Firstly, orders arrive bit by bit rendering the problem to a rolling or online one. Secondly customers placing orderings expect a promise concerning the (anticipated) day of delivery.

Selecting such a delivery date is not only a question of the available production capacities but also a question of achieving synergies within production processes in order to increase efficiency. Synergies can for example be achieved by properly sequencing or batching orders from different customers, e.g. in order to minimize setup costs. However, since orders arrive bit by bit, a certain amount of orders needs to be collected until synergies can finally be achieved.

To this end the future is divided into planning time intervals with increasing size towards the more distant future, reflecting decreasing planning granularity. Orders are then assigned to these planning time intervals and customers are promised delivery within the respective time interval. As time progresses larger planning time intervals are then gradually split into smaller ones. All orders within a larger time interval are then reassigned into the smaller planning time intervals and an updated delivery time interval is communicated. See Figure 1 for an example. A split is an offline problem where it is possible to precondition the scheduling problems to achieve synergies.

Fundamental prerequisite to this approach is the availability of a capacity oriented model which allows determining whether or not an order can still be executed within a time interval without solving a detailed scheduling problem. Detailed schedules are of course required for

Jens Leoff
Fraunhofer ITWM, Kaiserslautern, Germany
E-mail: jens.leoff@itwm.fraunhofer.de

Heiner Ackermann
Fraunhofer ITWM, Kaiserslautern, Germany
E-mail: heiner.ackermann@itwm.fraunhofer.de

Karl-Heinz Küfer
Fraunhofer ITWM, Kaiserslautern, Germany
E-mail: karl-heinz.kuefer@itwm.fraunhofer.de

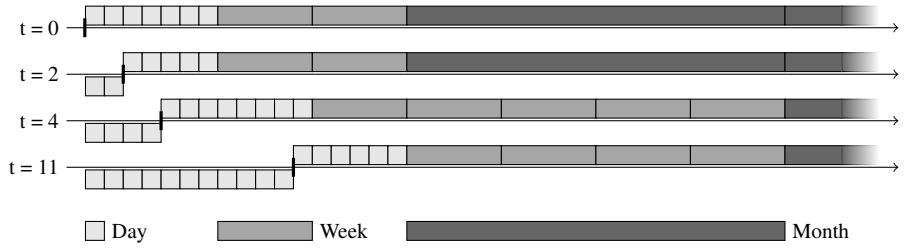


Fig. 1 Example of the rolling hierarchical scheduling model. While the near future is planned on a daily basis, the level of detail is reduced to month accuracy farther in the future. New orders are assigned to planning time intervals where they still fit. In order to always have a sufficiently large time horizon planned to the desired level of detail, intervals of the next larger size are split into a suitable amount of smaller intervals as time advances.

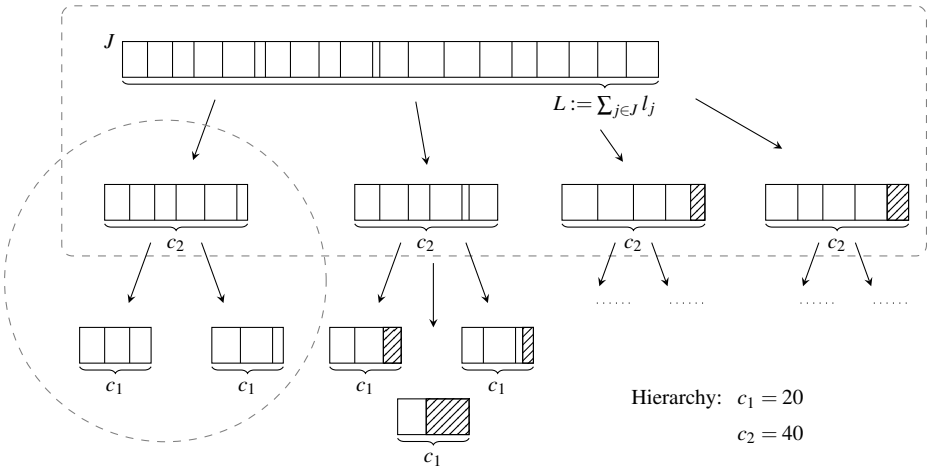


Fig. 2 Schematic view of a hierarchical packing. The set J of jobs in the picture has $|J| = 20$ and $\sum_{j\in J} l_j = 150$, with $(l_j)_{j\in J} = (7,7,6,8,9,3,7,8,6,9,2,8,10,10,9,7,9,7,9,9)$. The depicted hierarchical packing is minimal and needs 9 bins of size c_1. The optimal solution value to the Bin Packing Problem for J into bins of size c_1 is 8.

production needs, but they have to be computed only for the smallest sized time intervals in the near future.

In order to analyse such an approach mathematically, we first simplify as follows: We consider planning time intervals of different sizes as bins, such that bin sizes represent production capacities. This results in a hierarchy of different sized bins. We then solve a series of bin packing problems, where we first assign orders to bins of the largest size. Iteratively all the orders of a larger bin are reassigned to bins of the next smaller size, until all orders are assigned to bins of the smallest size. We call this procedure hierarchical bin packing. A hierarchical bin packing is called minimal if every assignment is an optimal bin packing solution. This approach might result in a suboptimal packing, i.e. directly packing orders in bins of smallest size might require less bins, see Figure 2 for an example.

We analyse how much solution quality is lost in the worst case using the hierarchic approach. To this end, we compare the number of smallest bins required by a hierarchical

bin packing to the number required by an optimal bin packing into bins of the smallest size directly.

Our findings are summarized as follows. We first conduct an offline analysis of the hierarchical bin packing problem and show that a minimal hierarchical packing requires at most $2h+1$ times more bins than an optimal packing. Here h denotes the height of the hierarchy, i.e. the number of different sizes of time intervals. We also present a lower bound of $h-1$ for the approximation factor.

Under mild assumptions on the sizes of the planning time intervals, this upper bound reduces to $3+\frac{2}{q-1}$ where q is greater than one and only depending on the sizes of the time intervals, i.e. we get a bound independent of h. We finally observe that the results immediately carry over to the corresponding online version of the problem, where new orders arrive in an online fashion.

Related Literature

The problem we are concerned with is integration of production planning and scheduling (IPPS) in make-to-order (MTO) environments. For an introduction to the importance and the problems in practice of integrated production planning and scheduling there are recent overview papers by Kreipl and Pinedo [2], Maravelias and Sung [4] and Gao et al. [3]. Our hierarchical approach fits well with the concepts of Hierarchical Production Planning suggested by Schneeweiss in [5].

Most models for IPPS we found in the literature consider the production planning part as a lot-sizing problem. However, this approach requires demand forecasts and is not appropriate for make-to-order production. We consider the planning problem as finding a due date assignment that positively preconditions the scheduling problem.

In [7], Zorzini et al. point out that for MTO companies the assignment of due dates while still considering production capacities is a key concern and they make an empirical study with 15 companies.

Due date assignment strategies are reviewed by Cheng and Gupta in [1]. The difference to our approach is that they do not allow a delay before deciding upon a precise due date. They do not consider rolling horizon or time interval models.

For more advanced models, aggregation of detailed scheduling constraints is essential for efficient estimation of the used capacity. Váncza et al. propose and analyse aggregation methods for precedence constraints in [6].

References

1. Cheng, T., Gupta, M.: Survey of scheduling research involving due date determination decisions. European Journal of Operational Research **38**(2), 156 – 166 (1989)
2. Kreipl, S., Pinedo, M.: Planning and scheduling in supply chains: An overview of issues in practice. Production and Operations Management **13**(1), 77–92 (2004)
3. Li, X., Gao, L., Zhang, C., Shao, X.: A review on integrated process planning and scheduling. International Journal of Manufacturing Research **5**(2), 161–180 (2010)
4. Maravelias, C.T., Sung, C.: Integration of production planning and scheduling: Overview, challenges and opportunities. Computers & Chemical Engineering **33**(12), 1919 – 1930 (2009)
5. Schneeweiss, C.: Distributed Decision Making. Springer (2003)
6. Váncza, J., Kis, T., Kovács, A.: Aggregation - the key to integrating production planning and scheduling. CIRP Annals - Manufacturing Technology **53**(1), 377 – 380 (2004)
7. Zorzini, M., Corti, D., Pozzetti, A.: Due date (dd) quotation and capacity planning in make-to-order companies: Results from an empirical analysis. International Journal of Production Economics **112**(2), 919 – 933 (2008)

MISTA 2013

Optimal scheduling of aircrafts' engines repair process

Jorge Orestes Cerdeira · Isabel Cristina Lopes · Eliana Costa e Silva

1 Introduction

We address a real world scheduling problem concerning the repair process of aircrafts' engines by TAP - Maintenance & Engineering (TAP-ME). TAP-ME is the maintenance, repair and overhaul organization of TAP Portugal, Portugal's leading airline, which employs about 4000 persons to provide maintenance and engineering services in aircraft, engines and components. TAP-ME is aiming to optimize its maintenance services, focusing on the reduction of the engines repair turnaround time.

When an engine arrives at the TAP-ME Lisbon's base, the engine is dismantled in mini-modules, and each mini-module is decomposed in smaller parts which are cleaned and checked for defects. If defects are identified, the part is allocated for repair and a repair process is established, composed of an ordered sequence of tasks which are fully registered in the repair ID label. Repair operations may consist of manual repair jobs, such as welding or screwing, or machine repairs, such as shot peening or oven heating, which are carried out in a number of workstations. Defective parts are taken across the plant through the necessary workstations in order to have the repairs done.

There are several workcenters at the Lisbon maintenance unit, such as SOLD - where the welding jobs are performed; PINT - for the painting jobs; FORNO - the vacuum furnace; REP - where all other general repairs are made.

Some of the specific jobs in the FORNO, Plasma Shot and Shot Peening workcenters require setup times, which are known at the beginning of the repair process of that part, and do not depend on the sequence of the tasks to be performed in that machine. Hence this information can be included in the ID label of the part, by adding the setup time to the processing time of each task. All groups allow preemption-resume, except the FORNO, Plasma Shot

Jorge Orestes Cerdeira
FCT-UNL and ISA/UTL
E-mail: orestes@isa.utl.pt

Isabel Cristina Lopes
ESEIG-IPP/CIEFGEI
E-mail: cristinalopes@eu.ipp.pt

Eliana Costa e Silva
UPT, ESTGF-IPP and ALGORITMI
E-mail: eliana@upt.pt

and Shot Peening. This means that in these three workcenters, once a job process is started, it cannot be interrupted. In all other machines, if a job is interrupted as a result of machine or worker unavailability, the job will continue processing after the machine or worker becomes available.

The process is carried in a dynamic environment, as the human resources can be transferred between different workstations, if necessary. During its repair process, an engine part can go around several workstations, and may even return to some of these workstations. At the end of the repair process, the parts are once again checked and the engine is reassembled.

TAP-ME aims to develop a model for the repair process to minimize the total tardiness, while considering different clients priority, i.e., discriminating between delays on the deadlines accorded for the repair and maintenance of engines from different clients.

2 The developed model

The model we present for the TAP-ME repair process of aircrafts' engines is a mixed integer linear programming (MILP) formulation for the flexible job shop scheduling problem [1] [2] [3], which combines linear ordering variables for sequencing the operations, with assignment variables, and linear variables for the starting times of the jobs.

We identify each part $P \in \mathscr{P}$, where $\mathscr{P}$ is the set of parts, as a sequence of operations $P = (o, o', ..., o^{(k)})$, and denote by A_P the set of pairs of consecutive operations of P. We let t_o be the processing time of operation o, O_e be the set of operations of parts of engine $e \in E$, where E is the set of engines, and let L be a large number. We use binary variables a_o^m to assign each operation $o \in O$ to exactly one processor $m \in M_o$, where M_o is the set of processors that can process job o; linear ordering variables $x_{oo'}$ to describe the precedence relationships between pairs of operations (o, o') that are performed on a same processor; and continuous variables to define the starting times S_o of the operations o and the tardiness T_e of each engine $e \in E$. The model adequates the MILP formulation [4] [5] of the classic job shop scheduling problem, to handle flexibility due to the fact that there are operations o that can be executed, on the same workstation, by any processor of the set of processors M_o. The objective function is defined as a weighted sum of the tardiness regarding due dates D_e for engine e, with a parameter $\lambda_e \geq 0$ to express the relative priority assigned to engine e.

$$
\begin{aligned}
\min \quad & \sum_{e \in E} \lambda_e T_e & & \\
\text{s.t.} \quad & \sum_{m \in M_o} a_o^m = 1, & & \forall o \in O \\
& S_{o'} \geq S_o + t_o, & & \forall (o, o') \in A_P, \forall P \in \mathscr{P} \\
& T_e \geq S_o + t_o - D_e, & & \forall o \in O_e, \forall e \in E \\
& x_{oo'} + x_{o'o} \geq a_o^m + a_{o'}^m - 1, & & \forall o \neq o' \in O, \forall m \in M_o \cap M_{o'} \\
& x_{oo'} + x_{o'o} \leq 1, & & \forall o \neq o' \in O : M_o \cap M_{o'} \neq \emptyset \\
& S_{o'} \geq S_o + t_o - L(1 - x_{oo'}), & & \forall o \neq o' \in O : M_o \cap M_{o'} \neq \emptyset \\
& S_o \geq S_{o'} + t_{o'} - L(1 - x_{o'o}), & & \forall o \neq o' \in O : M_o \cap M_{o'} \neq \emptyset \\
& a_o^m \in \{0, 1\}, & & \forall o \in O, m \in M_o \\
& x_{oo'} \in \{0, 1\}, & & \forall o \neq o' \in O : M_o \cap M_{o'} \neq \emptyset \\
& S_o \geq 0, & & \forall o \in O \\
& T_e \geq 0, & & \forall e \in E
\end{aligned}
$$

We can also accommodate in the proposed model the existence of non working periods and preemption-resume operations. Non working periods (e.g., lunch breaks, night rest breaks, non working days) can be easily handled defining, for each non working period of some processor (machine or worker) m, a dummy operation $d \in O_m$ with processing time t_d equal to the duration of the non working period, and assigning to variable S_d, the starting time of operation d, the starting time of the non working period.

3 Computational results

We experimented our model on data provided by TAP-ME[1] respecting an ordinary week at the Lisbon unit. The model was implemented in AMPL modelling language and solved using Gurobi 3.0.3. Figure 1 shows the results obtained with several instances extracted from the data provided for that week. We tested with all the available processors and with only half of the processors in each workstation. Larger instances were not solved due to memory limits (Intel Core i5-3450CPU@3.10GHz 8Gb RAM).

Problem						Model		Solution			
Instance	#engines	#jobs (n)	#operations	#worstations	#processors (m)	#variables	#constraints	Best objective	Gap	Iterations	CPU time (s)
Tap3M	3	22	132	11	96	5402	77198	0	0	218	1,17
Tap3M_W/2	3	22	132	11	52	4373	44894	0	0	198	0,58
Tap4M	4	24	141	11	96	6215	93575	0	0	245	1,43
Tap4M_W/2	4	24	141	11	52	5094	54315	0	0	238	0,64
Tap6M	6	110	283	11	96	27608	592531	0	0	417	18,70
Tap6M_W/2	6	110	283	11	52	25181	340835	0	0	477	68,09
Tap7M	7	130	394	11	96	46362	1032934	0	0	673	40,20
Tap7M_W/2	7	130	394	11	52	42976	594366	0	0	772	105,58
Tap8M	8	179	574	11	96	89749	2081327	179280	0	1261	261,63
Tap8M_W/2	8	179	574	11	52	84774	1198139	179280	(*)	(*)	(*)
Tap9M	9	215	810	11	96	157239	3673739	179280	(*)	(*)	(*)

(*) *Out of memory*

Fig. 1 Results of the MILP model for real instances.

For the sake of brevity, in Figure 2, we only show a fragment, concerning the first 480 minutes, of the solution found using real deadlines and priorities of the first instance. For example, PART10 undergoes operation "ILIMP_LIMPAR_PECA" on processor "ILIMP_13" from minute 0 to 60, followed by operation "IIDF_EFECT_DETECCAO_FENDAS" on processor "IIDF_1" from minute 60 to 120. We can also observe that on the workcenter REP at minute 0 start, simultaneously, three operations on three different processors (IREP_1, IREP_9, IREP_20). All parts of the three engines were processed long before the deadlines.

As referred above, we could not run the model for the whole week instance. For large instances, other approaches, including heuristic strategies, should be conceived. We plan to exploit this soon.

Acknowledgements We are grateful to TAP Maintenance & Engineering for having authorize us to report computational results on real data that they provided. The problem was initially presented at ESGI'86 held in Porto, May 2012. We thank Ana Bidarra and Francisco Correia, from TAP-ME, for their support during the ESGI'86 and in later contacts. We also would like to thank Manuel Cruz and Pedro Freitas of the ESGI86's organization for all the assistance.

[1] The id reference of the engines and parts, as well as the names of the repair tasks were modified, in order to respect our confidentiality agreement with TAP-ME.

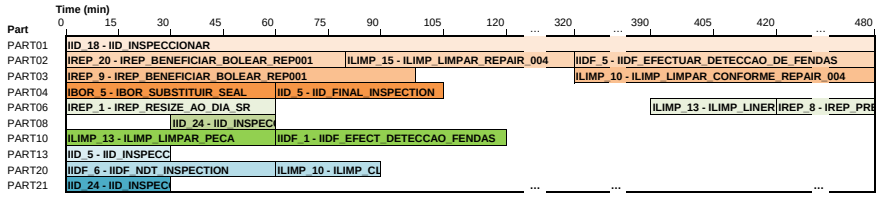


Fig. 2 Fragment of the solution, for the first 480 minutes, found for the three engines (MM1 - Parts 01 to 05; MM2 - Parts 06 to 11; MM3 - Parts 12 to 22). Each part is represented in a different color and it is indicated, for each period of time, the processor and the operation.

References

1. Brucker, P., Schlie, R.: Job-shop scheduling with multi-purpose machines. Computing **45**(4), 369–375 (1990)
2. Chan, F.T.S., Wong, T.C., Chan, L.Y.: Flexible job-shop scheduling problem under resource constraints. International Journal of Production Research **44**(11), 2071–2089 (2006)
3. Fattahi, P., Saidi Mehrabad, M., Jolai, F.: Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. Journal of Intelligent Manufacturing **18**(3), 331–342 (2007)
4. Manne, A.: On the job shop scheduling problem. Operations Research **8**, 219–223 (1960)
5. Unlu, Y., Mason, S.J.: Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. Computers and Industrial Engineering **58**(4), 785 – 800 (2010)

MISTA 2013

Optimal scheduling policies for diagnosing k-out-of-n systems with imperfect tests

Wenchao Wei · Fabrice Talla Nobibon · Roel Leus

1 Introduction

System health monitoring for complex systems, such as a space shuttle, aircraft or integrated circuits, is crucial for reducing the likelihood of accidents due to sudden failures, and for improving system availability. In this work, we study the development of optimal sequencing strategies for system testing. In particular, we look into k-out-of-n systems, which consist of n components. The systems function if at least k of the components are working [1]. We study the case where individual component tests are *imperfect*, which can involve two types of test errors and the terminology varies between references (compare with [2], for instance). Following natural parlance, we define the first type as *false positive*, which means the tester concludes the component is working ('up') when really it is not; for example, that a product fails a quality test before being shipped, while in reality it is in good condition. The second type of test error is *false negative*, which means the outcome is negative ('down') but in reality the component is up, for example, a system check-up failing to detect the fault it was designed to find, in a computer system that really has the fault.

In the reliability literature [3], the reliability of a system is the probability that the system functions. To evaluate the reliability, the assumption is that all components are examined; therefore the order in which the components are arranged for inspection does not influence system reliability or inspection costs. When the diagnostic procedure is actively monitored, however, sometimes not all components will need to be examined, and in this situation the sequencing of the tests does matter. Nachlas et al. [2], for instance, study the sequencing of imperfect component tests for series (n-out-of-n) systems. To the best of our knowledge, however, the more general sequencing problem of imperfect tests for k-out-of-n systems with $k \neq n$ has not yet been dealt with in the literature. It is the goal of this paper to fill exactly this gap. We assume that the diagnosis procedure halts once the test results (the output) satisfy a certain condition, namely that output accuracy would exceed a threshold value, and the objective is to design an optimal testing policy such that the total expected costs are minimized. One can imagine that under certain circumstances, testing the same component more than

Wenchao Wei, Fabrice Talla Nobibon, Roel Leus
ORSTAT, Faculty of Business and Economics, KU Leuven, Leuven, Belgium.
E-mail: {Wenchao.Wei, Fabrice.TallaNobibon, Roel.Leus}@kuleuven.be

once would improve the quality of the output (see [4], for example). In this paper, we do not consider such retesting: each component is tested at most once; Nachlas et al. [2] refer to this setting as 'single-pass' testing. This will allow us to focus only on the sequencing aspect.

2 Definitions

We consider a system consisting of n components, $U = \{u_1, u_2, \ldots, u_n\}$. The components should be tested sequentially to check whether they are in good condition or not. Each component is in one of two *states*: either working (up) or not working (down); component u_i is working with probability p_i, and we also define $q_i = 1 - p_i$. The system functions (succeeds) if at least k ($k \leq n$) out of the n components are working and malfunctions (fails) if at least $(n - k + 1)$ components are not working.

In order to discover the state of the system, we can test each component. The set of tests is denoted by $N = \{1, 2, \ldots, n\}$, where index i refers to the test of component u_i. The result of a component test is also binary: either positive or negative. We study the situation where the measurements (tests) are imperfect: a test can, for instance yield a positive outcome while the component is actually down, and vice versa. We assume that each test has a probability α of producing the wrong outcome. Specifically, for component u_i, if the test result is positive (negative), then we know the real state of the component is not-working (working) with a probability of α and working (not-working) with a probability of $(1 - \alpha)$. Such a common error probability for all components may be inherent in the design of the (unreliable) test machine, for instance. Each component is tested at most once. Tests are conducted one at a time; note that this is a dominant decision for our objective function (although this is not the case for other objectives such as makespan minimization).

We define an *outcome* as an n-dimensional binary vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{B}^n$, where $\mathbb{B} = \{0, 1\}$ and x_i denotes the result of test i, as follows: $x_i = 1$ if the result is positive and $x_i = 0$ if the result is negative. Let X_i represent the Bernoulli random variable with success probability χ_i and denote by $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ the associated vector of random variables before the start of testing procedure. The realization X_i is known after the test of component u_i. Let $\lambda_i = 1 - \chi_i$; we have $\chi_i(1 - \alpha) + \lambda_i \alpha = p_i$, and this leads to: $\chi_i = \frac{p_i - \alpha}{1 - 2\alpha}, \lambda_i = 1 - \chi_i = \frac{q_i - \alpha}{1 - 2\alpha}, i = 1, 2, \ldots, n$, where we require α to be sufficiently small, $0 < \alpha < \min\{p_i, q_i\} \leq 0.5$, such that χ_i and λ_i are positive.

A *schedule* $\mathbf{s} = (s_1, s_2, \cdots, s_{|\mathbf{s}|})$ is an ordered subset of N, in which s_t is the test of component u_{s_t}, which will be applied in the t^{th} position and $|\mathbf{s}|$ is the length of $\mathbf{s}$. Let Σ be the set of all schedules. Value c_i represents the cost of testing component u_i. We define cost function $f(\mathbf{s})$ for schedule $\mathbf{s}$ as $f(\mathbf{s}) = \sum_{t=1}^{|\mathbf{s}|} c_{s_t}$.

We will also consider the case in which precedence constraints apply between certain pairs of components. Specifically, a partial order relation (a transitive and irreflexive binary relation) on N is part of the input, implying that component i should be tested before j when $(i, j) \in A$.

3 Problem statement

In system testing, a decision rule uses diagnosis experience from test outcomes gathered over time. A solution to the sequencing problem under study is a *policy*, which consists

of a set of decision rules. At each stage, the policy shall mandate the next step to be taken: either conduct a new test, or stop; the stop/go decision when not all tests have been conducted yet is based on the accuracy regarding the system's state. After r tests have been applied, based on the outcomes, we conclude that the system functions with a probability $\theta_1(r, \mathbf{x})$ and malfunctions with a probability $\theta_0(r, \mathbf{x})$.

We study the optimization problem of designing a test policy Π with minimum expected total diagnostic costs. Given a specified threshold value T on system guarantee (e.g., $T = 95\%$), the inspection procedure stops when either $\theta_0(r, \mathbf{x})$ or $\theta_1(r, \mathbf{x})$ exceeds T; otherwise we shall perform all the n tests to maximize the accuracy of the result. In the context of isolating a single failure source, this guarantee is referred as a 'level of confidence'.

Following the literature on stochastic scheduling [5], a policy Π can also be seen as a function $\Pi : \mathbb{B}^n \to \Sigma$ that maps outcomes $\mathbf{x}$ to schedules. Note that several outcomes may be mapped to the same schedule. Our objective is to find a policy Π^* within a specific class that minimizes $\mathbb{E}[f(\Pi(\mathbf{X}))]$, with $\mathbb{E}[\cdot]$ the expectation operator with respect to $\mathbf{X}$. Specifically,

$$\mathbb{E}[f(\Pi(\mathbf{X}))] = \sum_{\mathbf{x}\in\mathbb{B}^n} \left(\prod_{i:x_i=1} \chi_i \right) \left(\prod_{i:x_i=0} \lambda_i \right) f(\Pi(\mathbf{x})). \tag{1}$$

An *elementary* policy is defined by a sequence $L = (l_1, l_2, \ldots, l_n)$ of all the tests, where l_t represents the test of component u_{l_t}, which is arranged in the t^{th} position in the sequence ($1 \leq t \leq n$). The diagnostic procedure executes the tests following the sequence produced by L one by one until a stop criterion is met. In the literature on system testing, this is typically called a *sequential* policy: the test sequence is selected before testing begins, as opposed to *non-sequential* testing, where the sequence may be adjusted based on the observed outcomes. A solution that has the minimum expected costs over all policies is called *globally* optimal. We find that even for special cases, such as $k = 1$ or $k = n$, elementary policies are not always globally optimal. The merit of an elementary policy is its compact representation, but it may be time-consuming to find an optimal one.

The problem without testing errors and with general precedence constraints can be shown to be NP-hard (based on [6]), and considering test errors does not make the problem easier. Producing a simple linear formulation for this problem is not straightforward: even for the special case $\alpha = 0$ and $k = 1$ or $k = n$, which admits a globally optimal elementary policy, a natural formulation would use variables $x_{ip} \in \{0; 1\}$ indicating whether component i is tested in position p but the objective function would not be linear. For general k values, even producing a compact formulation is not trivial. We therefore refrain from finding solutions by means of a MILP solver and we develop a dynamic programming algorithm to find globally optimal solutions. In our computational results, we will map the effect on the total costs of introducing test errors.

References

1. R. Butterworth, Some reliability fault-testing models, *Operations Research*, 20, 335-343, 1972
2. J. Nachlas, S. Loney and B. Binney, Diagnostic-strategy selection for series systems, *IEEE Transactions on Reliability*, 39, 273-280, 1990

3. S. V. Amari, H. Pham and G. Dill, Optimal design of k-out-of-n:G subsystems subject to imperfect fault-coverage, *IEEE Transactions on Reliability*, 53, 567-575, 2004
4. J. Ding, B. Greenberg and H. Matsuo, Repetitive testing strategies when the testing process is imperfect, *Management Science*, 44, 1367-1378, 1998
5. G. Igelmund, and F. Radermacher, Preselective strategies for the optimization of stochastic project networks under resource constraints, *Networks*, 13, 1-28, 1983
6. De Reyck, B. and R. Leus, R&D-project scheduling when activities may fail, *IIE Transactions*, 40, 367-384, 2008

MISTA 2013 manuscript No.
(will be inserted by the editor)

Student Sectioning at High Schools in Denmark

Simon Kristiansen · Andrew Mason · Thomas R. Stidsen

the date of receipt and acceptance should be inserted later

1 Introduction

High School Student Sectioning (HSSS) is a planning problem each Danish high school faces every year. There has been an increase in scientific research within Education Timetabling and High School Timetabling during the last decade, perhaps as a result of the three International Timetabling Competitions. However, much of the research is concerned with the actual timetabling problem and not the sectioning of students. The HSSS problem is closely related to the Elective Course Student Sectioning (ECSS) problem (Kristiansen et al (2011)). Both problems are yearly recurrent planning problems at Danish high schools. However, while the ECSS concentrates on assigning 2nd and 3rd year students to elective courses given their requests, the HSSS is concerned with the partitioning of 1st year students to partitioning the students into "cohorts" that will be timetabled as one group for most, but not all, courses. The overall goal of this work is to create a solution method which can be implemented in the cloud based high school administration software Lectio and hence we can make available to the majority of Danish high schools. This paper introduces IP models for the problem and reports solution performance using the state-of-art MIP solver Gurobi.

2 High School Student Sectioning

The HSSS problem can be treated as two sub-problems, the first being to partition students into student cohorts and the second being to create timetables for these

S. Kristiansen (E-mail: sikr@dtu.dk) · T.R. Stidsen
Section of Operations Research, Department of Management Engineering, Technical University of Denmark
Building 426, Produktionstorvet, DK-2800 Kgs. Lyngby, Denmark

S. Kristiansen
MaCom A/S, Vesterbrogade 48, 1., DK-1620 Copenhagen V, Denmark

A. Mason
Dept. of Engineering Science, University of Auckland, Auckland, New Zealand

cohorts. While these problems are interconnected, we implement a fast heuristic approach in which we solve the subproblems sequentially. The first sub-problem is the partitioning of students to form "cohorts", where a cohort of students will be timetabled (in the second sub-problem) as one entity for most of their classes. In our problem, students are distinguished by their choice of specialization, where some possible specializations are listed in Table 1. Each student chooses exactly one specialization. In this example, the school has specified that we need to form 5 cohorts each containing no more than 30 students where each cohort contains students from one or more specializations. We wish to minimize the sum, over all cohorts, of the number of specializations in each cohort. Table 1 shows a possible solution with a specialization sum of 7 over the 5 cohorts.

Table 1 An example of forming cohorts for linguistic students who have chosen one of 4 linguistic specializations. The upper limit on cohort size is 30 students

Specialization	# Students	Cohort 1	Cohort 2	Cohort 3	Cohort 4	Cohort 5
English-German	18		18			
English-Spanish	40	28	12			
English-Music	9			9		
English-SocialSci	77			20	30	27

The example above considered students each of whom has chosen a specialization in the linguistic area. Students who have chosen not to specialize in the linguistic will instead have chosen a specialization from another list of science specialization. A similar independent problem needs to be solved to form cohorts for these students.

As well as choosing either an linguistic or science-specialization, all students chooses one Linguistic elective, such as French or Italian, and one Artistic elective such as Drama or Media. While each cohort is treated as a single entity for their specialization teaching, this is not so for the electives where students from many cohorts may be combined to form an elective class. Each cohort must be assigned a single schedules time-slot for their Linguistic elective, and similarly must have a single scheduled time-slot for their Artistic elective. Thus, after partitioning students into cohorts, we wish to determine for each cohort the time-slot that will be used for their Linguistic elective and the time-slot that will be used for their Artistic elective. These assignments then determine the number of students taking each elective, such as French or Italian, in each time-slot, and elective classes can then be created based on these numbers. An efficient solution will typically have many cohorts assigned to the same time-slot, reducing the number of classes that need to be run for each elective.

The primary objective of the HSSS is to assign all the students to a cohort, while minimizing the number of specializations represented in each cohort. As a second objective, we want to minimize the amount of elective classes needed to fulfill all elective requests.

2.1 IP model

The consider the first problem to be a fixed charge minimum cost flow problem, where the flow is the amount of students partitioned into a cohort. For the partitioning of students into cohorts at a high school we have a set of students, a set of specializations, $s \in \mathcal{S}$, and a set of cohorts , $c \in \mathcal{C}$. We introduce the dummy cohort $c_\emptyset$. The flow going in to the dummy cohort denotes the students which cannot be placed in a cohort due to the restrictions. To avoid some of the symmetry which the problem contains, we introduce the set of student groups, $g \in \mathcal{G}$. A student group consists of students which have requested exactly the same specialization and elective. Let $L \in \mathbb{N}$ and $U \in \mathbb{N}$ be the lower and upper limit on the cohort size specified by the schools, respectively. Let $F_g \in \mathcal{N}$ be the size of the student group g, and let $A_{g,g'} \in \{0,1\}$ be 1 if the student groups g and g' are not allowed in the same cohort. The decision variable $x_{g,c} \in \{0,1\}$ takes value 1 if any flow from student group g to cohort c, and zero otherwise. The variable $\bar{x}_{g,c} \in \mathbb{N}$ is the number of student from student group g assigned to cohort c. Furthermore, to ease the teaching of mandatory courses we wish to have as few specializations and electives represented in each cohort. Let the parameter $R_{g,e} \in \{0,1\}$ be 1 if student group g has requested elective e, and zero otherwise and let $D_{g,s} \in \{0,1\}$ take value 1 if student group g has requested specialization s. The variables $z_{s,c}$ $\in \{0,1\}$ and $z_{e,c} \in \{0,1\}$ denote whether specialization s or course e is represented in cohort c, respectively, and zero otherwise.

The subproblem of forming cohorts can now be written as follows.

$$\min \alpha \cdot \sum_{g\in\mathcal{G},c\in\mathcal{C}} x_{g,c} + \beta \cdot \sum_{g\in\mathcal{G}} \bar{x}_{g,c_\emptyset} + \sum_{s\in\mathcal{S},c\in\mathcal{C}} \delta_s \cdot z_{s,c} + \sum_{e\in\mathcal{E},c\in\mathcal{C}} \delta_e \cdot z_{e,c} \quad (1)$$

$$\text{s.t.} \sum_{c\in\mathcal{C}} \bar{x}_{g,c} + \bar{x}_{g,c_\emptyset} = F_g \qquad \forall g \in \mathcal{G} \quad (2)$$

$$\bar{x}_{g,c} \leq \min[F_g, U] \cdot x_{g,c} \quad \forall g \in \mathcal{G}, c \in \mathcal{C} \quad (3)$$

$$\sum_{g\in\mathcal{G}} \bar{x}_{g,c} \leq U \qquad \forall c \in \mathcal{C} \quad (4)$$

$$\sum_{g\in\mathcal{G}} \bar{x}_{g,c} \geq L \qquad \forall c \in \mathcal{C} \quad (5)$$

$$x_{g,c} + x_{g',c} \leq 1 \qquad \forall c \in \mathcal{C}, g, g' \in \mathcal{G}, g \neq g' A_{g,g'} = 1 \quad (6)$$

$$D_{g,s} \cdot x_{g,c} \leq z_{s,c} \qquad \forall g \in \mathcal{G}, c \in \mathcal{C}, s \in \mathcal{S} \quad (7)$$

$$R_{g,e} \cdot x_{g,c} \leq z_{e,c} \qquad \forall g \in \mathcal{G}, e \in \mathcal{E}, c \in \mathcal{C} \quad (8)$$

$$x_{g,c} \in \{0,1\} \quad (9)$$

$$\bar{x}_{g,c} \in \mathcal{N} \quad (10)$$

$$z_{s,c} \in \{0,1\} \quad (11)$$

$$z_{e,c} \in \{0,1\} \quad (12)$$

Constraints (2) ensure that all students in a student group are assigned a cohort. Constraints (3) ensure that the number of students from student group g into cohort c, cannot be greater that the upper cohort size or the size of the student group. Constraints (4) and (5) make sure that the total flow into a cohort is limited by the upper and lower cohort size. Constraints (6) make sure that two student groups which are not allowed to be placed in the same cohort are not done so. Finally constraints (7) and (8) sets the value for the variables $z_{s,c}$ and $z_{e,c}$ and

hence denote if specialization s or course e is represented in cohort c, respectively.

The second part of the HSSS is the elective timetabling problem where we want to assign each cohort to one time-slot for their Linguistic elective and one time-slot for their Artistic elective. We take as input the cohorts for all students produced by solving the first sub-problem. Let $T_d \subseteq \mathcal{T}$ be the set of times available for electives of type d where $d \in \mathcal{D} = \{\text{Linguistic, Artistic}\}$, and $\mathcal{T}$ is set of all elective time slots. The decision variable $y_{c,t,d} \in \{0,1\}$ denotes whether cohort c is assigned time-slot t for elective type d. We let variable $\bar{v}_{e,t} \in \mathcal{N}$ be the number of classes needed for elective e in time slot t and variable $v_{e,t} \in \{0,1\}$ take value 1 if elective e is presented in time slot t. The parameter $K_{e,t} \in \mathcal{N}$ denotes the maximum number of classes of elective e which can be created in time slot t, where one class in elective e can take up to U_e students. The objective of this sub-problem is to minimize the number of elective classes required and to have as few electives represented in each time, which we achieve using the following model.

$$\min \sum_{e\in\mathcal{E},t\in\mathcal{T}} \bar{v}_{e,t} + \sum_{e\in\mathcal{E},t\in\mathcal{T}} \omega \cdot v_{e,t} \tag{13}$$

$$\text{s.t.} \sum_{t\in T_d} y_{c,t,d} = 1 \qquad \forall c \in \mathcal{C}, d \in \mathcal{D} \tag{14}$$

$$\sum_{c\in\mathcal{C}} y_{c,t,d} \geq 1 \qquad \forall t \in \mathcal{T}, d \in \mathcal{D} \tag{15}$$

$$\sum_{g\in\mathcal{G}} R_{g,e} \cdot \bar{x}_{g,c} - M(1 - y_{c,t,d}) \leq U_e \cdot \bar{v}_{e,t} \quad \forall c \in \mathcal{C}, e \in \mathcal{E}, t \in \mathcal{T}, d \in \mathcal{D} \tag{16}$$

$$\bar{v}_{e,t} \leq K_{e,t} \qquad \forall e \in \mathcal{E}, t \in \mathcal{T}, K_{e,t} > 0 \tag{17}$$

$$z_{e,c} + y_{c,t,d} - 1 \leq v_{e,t} \qquad \forall e \in \mathcal{E}, c \in \mathcal{C}, d \in \mathcal{D} \tag{18}$$

$$y_{c,t,d} \in \{0,1\} \tag{19}$$

$$\bar{v}_{e,t} \in \mathcal{N} \tag{20}$$

$$v_{e,t} \in \{0,1\} \tag{21}$$

Constraints (14) make sure that every cohort is assigned exactly one time slot for each course type. Constraints (15) ensure that there is at least one cohort in each time slot, whereas Constraints (16) sets the value for variable $\bar{v}_{e,t}$. Constraints (17) ensure that the recourse limitation for a given course at a given time is respected. Constraints (18) denote whether elective e is represented in time slot t, using variables $v_{e,t}$.

3 Solution Method and outlook

Currently the model has been tested using Gurobi using 15 real life datasets provided by MaCom A/S, the owner of Lectio.The results shows the sequential heuristic in average performs better than solving the entire model using a running time of 60 seconds. Work is continuing on refining these to develop an implementation of a solver for HSSS in Lectio.

References

Kristiansen S, Srensen M, Stidsen TR (2011) Elective course planning. European Journal of Operational Research 215(3):713 – 720, DOI 10.1016/j.ejor.2011.06.039

MISTA 2013

Algorithms for building optimal aircraft maintenance personnel rosters

Philippe De Bruecker · Erik Demeulemeester · Jeroen Beliën · Jorne Van den Bergh

1 Introduction

Scheduling aircraft maintenance personnel at an aircraft maintenance company entails some special problems [1,2]. First, the workforce scheduling problem is heavily constrained by labour union agreements. Second, the maintenance jobs that have to be carried out are malleable. This means that the company should decide when exactly the maintenance will take place between the arrival and departure of the aircraft. Hence, the timing of the workload is an extra decision in the scheduling problem. Third, aircraft tend to not always fly on schedule and arrive now and then with a delay. When the workforce scheduling does not anticipate delays in arrival time, the scheduled capacity may be insufficient to maintain all aircraft in time.

This extended abstract presents an overview of our current research on algorithms for building optimal aircraft maintenance personnel rosters. We distinguish between two different scenarios. In the first scenario, all flights arrive exactly on time according to the flight schedule. For this deterministic scenario, two subsequent Tabu Search algorithms are proposed to minimize the total labour costs and to maximize employee satisfaction.

In the second scenario, we recognize that aircraft do not always fly on schedule and sometimes arrive with a delay. To cope with stochastic aircraft arrival times, a Model Enhancement technique is proposed to obtain robust aircraft maintenance personnel rosters that minimize the total labour costs. We define the stochastic robustness of these rosters as their ability to ensure a certain service level; i.e., to ensure that on average at least a certain percentage of the flights can be maintained before their Scheduled Time of Departure (STD).

This research is inspired by the real life problems at Sabena Technics, a large aircraft maintenance company located at Brussels Airport in Belgium.

Philippe De Bruecker
KULeuven
E-mail: philippe.debruecker@kuleuven.be

Erik Demeulemeester
KULeuven
E-mail: erik.demeulemeester@kuleuven.be

Jeroen Beliën
HUBrussels
E-mail: jeroen.belien@hubrussel.be

Jorne Van den Bergh
HUBrussels
E-mail: jorne.vandenbergh@hubrussel.be

2 Problem definition

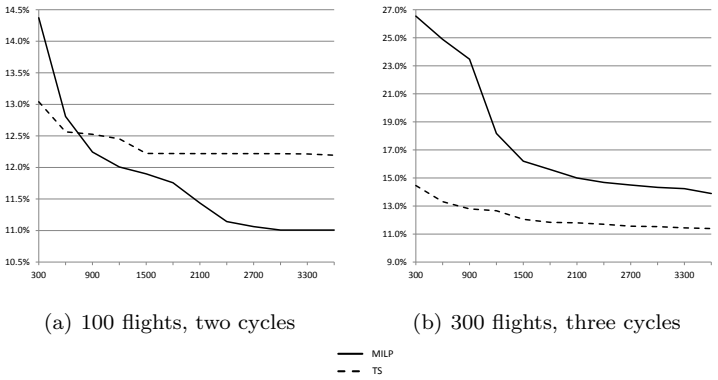
At Sabena Technics, the demand pattern for aircraft maintenance repeats every week and changes only twice a year. Given such a recurring workload for each flight during one week, we have to find the optimal workforce configuration in terms of the lowest total labour costs for the next season.

The maintenance personnel at Sabena Technics works in teams of a certain team size. Each team can only work one shift per day or have a day off. There are four possible shift types with overlapping working hours. This results in a certain sequence of shift types and days off assigned to a certain team for a certain week. A feasible workforce configuration must be able to cover the demand for services at all times. Real life staffing and scheduling problems are not limited to finding the lowest possible costs while satisfying the demand for their services. They must also follow strict union regulations. Such regulations concerning average working times, working in the weekend, break times, shift succession constraints, etc., are extra constraints in the optimization model.

3 Tabu Search for the deterministic scenario

In order to build optimal aircraft maintenance personnel rosters, the total labour costs should be minimized without neglecting the employee satisfaction. Hence, besides the cost minimization, maximizing employee satisfaction is a second goal. These two goals are defined as the objective functions of two subsequent Tabu Search optimization models. The only link between these models is that the second model (the satisfaction model) uses the output from the first model (the cost model) as its input. To enhance the performance of the algorithm, strategic oscillation and pattern creation are used to cope with the many labour union constraints.

Fig. 1 Comparison of the evolution of the average optimality gaps in two scenarios



(a) 100 flights, two cycles (b) 300 flights, three cycles

The gaps (as percentages) are presented on the vertical axes. The computation times (in seconds) are presented on the horizontal axes.

We illustrate the performance of our Tabu Search (TS) approach with a computational experiment and compare the results to a mixed integer linear programming (MILP) model. Figure 1 presents the results of the cost model in terms of the average

optimality gaps over time for the TS and MILP approach in two different scenarios. In the first scenario, only 100 flights and two cyclic workforce schedules are considered. The second scenario considers a larger and more realistic problem instance with 300 flights. In this case, the goal is to construct a workforce schedule consisting of three cycles. The experiments demonstrate that Tabu Search can deliver acceptable solutions faster than the integer programming technique for small instances and outperforms the integer programming model for larger and more realistic problem sizes.

4 Model Enhancement for the stochastic scenario

Unfortunately, aircraft do not always fly on schedule and sometimes arrive with a delay. Unexpected delays can disarrange the maintenance planning which can lead to a delay in the maintenance of aircraft when capacity buffers are absent or too small. To ensure a high level of service to its customers in such a stochastic environment, Sabena Technics allows for at most a certain average percentage of the flights to be maintained after their scheduled time of departure.

Several techniques have already been proposed in the literature to cope with the problems of stochastic optimization. However, no single best solution method exists. The most simple technique is to approach the stochastic parameters in the optimization model by their averages and use deterministic solution techniques. Bachelet and Yon (2007) refer to this as straightforward optimization (SFO). For more complex situations, simulation is often used.

According to Bachelet and Yon (2007), the techniques that use simulation for stochastic optimization can be classified in two categories: simulation optimization and model enhancement (ME). Simulation optimization works by evaluating different solutions with simulation and usually guides the search through different solutions to obtain a good (or the best) feasible result [4]. The term "model enhancement" is used to indicate a different way of combining simulation and optimization. While most optimization-simulation couplings focus on improving the objective function evaluated from simulation (like the simulation optimization approach), ME still focuses on optimizing the theoretical objective function. ME tries to improve the solution provided by a mathematical model by the use of simulation.

For our research, we developed such a model enhancement procedure that enhances a Mixed Integer Linear Programming (MILP) model to solve the stochastic problem at Sabena Technics and compare the results with SFO. The experiments demonstrate that the ME approach always succeeds in finding a feasible solution (that satisfies the stochastic constraint) with lower costs compared to different SFO models.

References

1. Beliën, J., Demeulemeester, E. & Cardoen, B., Improving workforce scheduling of aircraft line maintenance at Sabena Technics, Interfaces, 42 (4), 352-364 (2012)
2. Beliën, J., De Bruecker, P., Demeulemeester, E., Cardoen, B. & Van den Bergh, J., Integrated staffing and scheduling for an aircraft line maintenance problem, Computers & Operations Research, 40 (4), 1023-1033 (2013)
3. Bachelet, B. & Yon, L., Model enhancement: Improving theoretical optimization with simulation, Simulation Modelling Practice and Theory, 15 (6), 703-715 (2007)
4. Chen, C.-H. & Lee, L.H., Stochastic Simulation optimization, World Scientific, Singapore (2011)

MISTA 2013

SOLUTION ALGORITHMS FOR THE TWO-STATION SINGLE TRACK RAILWAY SCHEDULING PROBLEM

Evgeny R. Gafarov · Alexandre Dolgui · Alexander A. Lazarev

1 Introduction

A single track network can be seen as an embryonic portion for any type of railway network topology. In this paper, we consider a special **core** subcase with only two stations. This subcase has practical significance and appears often in private railways, e.g., when a company transports loads between two production centers.

The Single Track Railway Scheduling Problem with two stations (STRSP2) is formulated as follows. Given a single track railway between two stations and a set $N' = N'_1 \bigcup N'_2,\ N'_1 \bigcap N'_2 = \emptyset$ of $n' = |N'|$ trains. Trains from the subset N'_1 go from station 1 to station 2, and trains from the subset N'_2 go in the opposite direction. $|N'_1| = n'_1$ and $|N'_2| = n'_2,\ n'_1 + n'_2 = n'$. The track is divided on Q segments $1, 2, \ldots, Q$. Trains from the set N'_1 traverse segments in an order $1 \to 2 \to \ldots \to Q$ and trains from the set N'_2 in the opposite order $Q \to Q-1 \to \ldots \to 1$. At most only one train can be on any track segment at a time[1]. If a train $j' \in N'_1$ is on a track segment, then no train $i' \in N'_2$ can be on the track and vice versa. For each segment $q,\ q = 1, 2 \ldots, Q$, a traversing time p_q is given, in which a train $j' \in N'$ traverses the segment, i.e., for each segment $q,\ q = 1, 2 \ldots, Q$, all the trains go with the same speed[2]. Let $S_{j'}(\Pi)$ and

E.R. Gafarov
Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya st. 65, 117997 Moscow, Russia
E-mail: axel73@mail.ru

A.Dolgui
Ecole Nationale Superieure des Mines, CNRS UMR6158, LIMOS, F-42023 Saint-Etienne, France
E-mail: dolgui@emse.fr

A.A. Lazarev
Lomonosov Moscow State University, Higher School of Economics (National Research University), Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya st. 65, 117997 Moscow, Russia,
E-mail: jobmath@mail.ru

[1] A segment is circumscribed by two signals: one signal from each side, which will control when a train either can or cannot proceed on that segment. This is a safety precaution. So, there is no opportunity for trains to pass each other somewhere between segments.

[2] This assumption is not far away from practice, since most trains travel at a maximal speed allowed.

$C_{j'}(\Pi)$, $j' \in N'$, be the start and completion times of the train j' in a schedule Π, i.e., $S_{j'}(\Pi)$ is a departure time of the train j' from the departure station and $C_{j'}(\Pi)$ is an arrival time to the destination station. Then in a feasible schedule we have:

- $C_{j'} \geq S_{j'} + \sum_{q=1}^{Q} p_q$, $\forall j' \in N'$;
- for any $i' \in N_1'$ and for any $j' \in N_2'$ we have $C_{i'} \leq S_{j'}$ or $C_{j'} \leq S_{i'}$.

In addition, a due date $d_{j'}$, a weight $w_{j'} > 0$, a release date $r_{j'} \geq 0$ (the earliest possible starting time, i.e., $S_{j'} \geq r_{j'}$) for each train $j' \in N'$ can be given. If $C_{j'}(\Pi) > d_{j'}$, then train j' is tardy and we have $U_{j'}(\Pi) = 1$. If $C_{j'}(\Pi) \leq d_{j'}$, then train j' is on-time and $U_{j'}(\Pi) = 0$. Moreover, let $T_{j'}(\Pi) = \max\{0, C_{j'}(\Pi) - d_{j'}\}$ be the tardiness of train j' and $C_{max}(\Pi) = \max_{j' \in N'}\{C_{j'}(\Pi)\}$ is the makespan for schedule Π. For the STRSP2 with release dates the objective is to find an optimal schedule Π^* that minimizes the makespan C_{max} taking into account release dates. This problem is denoted $STRSP2|r_j|C_{max}$ (similar to the traditional three-field notation $\alpha|\beta|\gamma$ for scheduling problems proposed by Graham et al. [2], where α describes the resource environment, β gives the activity characteristics and further constraints and γ describes the objective function). In this paper, we deal with some extensions of STRSP2 with different objective functions and further constraints. We minimize

- number of late trains $STRSP2||\sum U_j$;
- weighted number of late trains $STRSP2||\sum w_j U_j$;
- total completion time $STRSP2|r_j|\sum C_j$ when release dates are given;
- weighted total completion time $STRSP2||\sum w_j C_j$;
- total tardiness $STRSP2||\sum T_j$.

Similar problems arise on a river canal (inland waterways) with a chain of shipping locks [1]. Although, this problem seems to be similar to STRSP2, there are obvious differences between the problems.

To the best of our knowledge there are no publications for this set of STRSP2 problems, although they can be also easily reformulated as shop scheduling problems with Q machines. A literature review on the single track railway scheduling problem can be found, e.g., in [4].

2 Reduction of STRSP2 to a Single Machine Scheduling Problem

Denote $p_{max} = \max_{q=1,2,\ldots,Q}\{p_q\}$ and $P = \sum_{q=1}^{Q} p_q$.

Lemma 1 *Assume that for a train $j' \in N_1'$ we have $C_{j'} = S_{j'} + P$ and train $i' \in N_1'$ is the next train which passes the track. Then, without violation of feasibility conditions the train i' can be scheduled as follows: $S_{i'} = \max\{r_{i'}, S_{j'} + p_{max}\}$ and $C_{i'} = S_{i'} + P$, i.e., the train i' departs from the time point $\max\{r_{i'}, S_{j'} + p_{max}\}$ and leaves without incurring any idle-time.*

Lemma 2 *For any j' and i' belong to the same subset N_1' or N_2', in any feasible schedule, we have $|S_{j'} - S_{i'}| \geq p_{max}$ and $|C_{j'} - C_{i'}| \geq p_{max}$.*

Based on these properties, the following reduction to a single machine scheduling problem is proposed.

Single machine scheduling problem A set $N = N_1 \bigcup N_2,\ N_1 \bigcap N_2 = \emptyset$ of n independent jobs that must be processed on a single machine is given. Job preemption is not allowed. The machine can handle only one job at a time. Processing times of jobs are equal to $p,\ \forall j \in N$. For each job $j \in N$, a due date d_j, a weight $w_j > 0$, a release date $r_j \geq 0$ (i.e., the earliest possible starting time) can be given. A feasible solution is described by a permutation $\pi = (j_1, j_2, \ldots, j_n)$ of the jobs of the set N from which the corresponding schedule can be uniquely determined by starting each job as early as possible. Let $S_{j_k}(\pi),\ C_{j_k}(\pi) = S_{j_k}(\pi) + p$ be the start and completion times of job j_k in the schedule resulting from the sequence π. If $j_k \in N_1$ and $j_{k+1} \in N_2$, then between jobs the machine has to be idle during a setup time $st = st_1$. If $j_k \in N_2$ and $j_{k+1} \in N_1$, then between jobs the machine has to be idle during a setup time $st = st_2$. There is no setup time between processing of jobs from the same subset, i.e., $st = 0$. In a feasible schedule $S_{j_{k+1}} = \max\{r_{j_{k+1}}, C_{j_k} + st\}$ holds. Objective functions are the same like for $STRSP2$. If $C_j(\pi) > d_j$, then job j is tardy and we have $U_j(\pi) = 1$, otherwise $U_j(\pi) = 0$. If $C_j(\pi) \leq d_j$, then job j is on-time. Moreover, let $T_j(\pi) = \max\{0, C_j(\pi) - d_j\}$ be the tardiness of job j and $C_{max}(\pi) = \max_{j \in N}\{C_j(\pi)\}$ is the makespan. We note these scheduling problems according to the traditional three-field notation $\alpha|\beta|\gamma$, e.g., $1|setup - times, N_1, N_2, p_j = p, r_j|C_{max}$ for the single machine scheduling problem with equal-processing-times, setup times and release dates minimizing makespan.

The problems $STRSP2|-|-$ for the previously mentioned objective functions can be reduced to $1|setup-times, N_1, N_2, p_j = p, -|-$ problems as follows. Subset of trains N_1' corresponds to the subset of jobs $N_1,\ |N_1| = |N_1'|$, and subset N_2' of trains to the subset $N_2,\ |N_2| = |N_2'|$, of jobs. Let $q,\ q \in \{1, 2, \ldots, Q\}$ be the index of segment for which $p_q = p_{max}$. Denote $TAIL_{left} = \sum_{l=1}^{q-1} p_l,\ TAIL_{right} = \sum_{l=q+1}^{Q} p_l$. Then, assume $p = p_{max},\ st_1 = 2 \cdot TAIL_{right},\ st_2 = 2 \cdot TAIL_{left}$, if $j \in N_1$, then release date $r_j = r_{j'} + TAIL_{left}$, else $r_j = r_{j'} + TAIL_{right}$. If $j \in N_1$, then due date $d_j = d_{j'} - TAIL_{right}$, else $d_j = d_{j'} - TAIL_{left}$. Weights are the same.

A similar reduction can be made for other problem. Thus, instead of STRSP2 the following $1|setup - times, N_1, N_2, p_j = p, -|-$ problems can be considered:

1. $1|setup - times, N_1, N_2, p_j = p, r_j|C_{max}$;
2. $1|setup - times, N_1, N_2, p_j = p, r_j|\sum C_j$;
3. $1|setup - times, N_1, N_2, p_j = p|\sum w_j C_j$;
4. $1|setup - times, N_1, N_2, p_j = p|\sum T_j$;
5. $1|setup - times, N_1, N_2, p_j = p|\sum U_j$;
6. $1|setup - times, N_1, N_2, p_j = p|\sum w_j U_j$.

Some results in equal-processing-time scheduling are presented in [3].

Definition 1. We call schedules for $1|setup - times, N_1, N_2, p_j = p, -|-$ problems *left-shifted*, if they are determined by starting each job as early as possible. Obviously, for any afore mentioned problem there are optimal schedules which are left-shifted.

Definition 2. Let $\Theta = \{t | t = r_j + x_1 \cdot p + x_2 \cdot st_1 + x_3 \cdot st_2,\ j \in \{1, 2, \ldots, n\},\ x_1, x_2, x_3 \in \{0, 1, 2, \ldots, n\},\ x_2 + x_3 \leq x_1\}$.

Notice that there are at most $O(n^4)$ values in set Θ.

Lemma 3 *In all left-shifted schedules, job starting times belong to* Θ.

3 Algorithms for the Problems with Ordered Subsets N_1 and N_2

Lemma 4 *Problems 1-4 are solvable in* $O(n^7)$ *or in* $O(n^6)$ *time.*

All the algorithms are based on the same properties of optimal solutions and use the same search procedure.

Denote the subset $N_1 = \{j_1, j_2, \ldots, j_{n_1}\}$ and $N_2 = \{i_1, i_2, \ldots, i_{n_2}\}$.

Lemma 5 *For each of the above mentioned problems there is an optimal schedule in which jobs are processed in the following special order:*

- *for the problems* $1|setup{-}times, N_1, N_2, p_j = p, r_j|C_{max}$ *and* $1|setup{-}times, N_1, N_2, p_j = p, r_j|\sum C_j$ *jobs are ordered according to non-decreasing release dates, i.e.,* $r_{j_1} \leq r_{j_2} \leq \ldots \leq r_{j_{n_1}}$ *and* $r_{i_1} \leq r_{i_2} \leq \ldots \leq r_{i_{n_2}}$*;*
- *for the problem* $1|setup - times, N_1, N_2, p_j = p|\sum w_j C_j$ *jobs in each subset are ordered according to non-increasing weights, i.e.,* $w_{j_1} \geq w_{j_2} \geq \ldots \geq w_{j_{n_1}}$ *and* $w_{i_1} \geq w_{i_2} \geq \ldots \geq w_{i_{n_2}}$*;*
- *for the problem* $1|setup - times, N_1, N_2, p_j = p|\sum T_j$ *jobs in each subset are ordered according to non-decreasing due dates, i.e.,* $d_{j_1} \leq d_{j_2} \leq \ldots \leq d_{j_{n_1}}$ *and* $d_{i_1} \leq d_{i_2} \leq \ldots \leq d_{i_{n_2}}$*.*

4 Problems with Partially Ordered Subsets

Lemma 6 *For the problem* $1|setup - times, N_1, N_2, p_j = p|\sum w_j U_j$*, there is an optimal left-shifted schedule, where on-time jobs from the same subset* N_1 *or* N_2 *are ordered according to non-decreasing due dates, i.e.,* $d_{j_1} \leq d_{j_2} \leq \ldots \leq d_{j_{n_1}}$ *and* $d_{i_1} \leq d_{i_2} \leq \ldots \leq d_{i_{n_2}}$*.*

Lemma 7 *Assume, that the jobs are ordered according to Lemma 6. For the problem* $1|setup{-}times, N_1, N_2, p_j = p|\sum U_j$*, there is an optimal left-shifted schedule and such indexes* $x,\ 1 \leq x \leq n_1$ *and* $y, 1 \leq y \leq n_2$*, that only jobs* $j_x, j_{x+1}, \ldots, j_{n_1}, i_y, i_{y+1}, \ldots, i_{n_2}$ *are on-time and processed according to the order given by Lemma 6.*

So, for the problem $1|setup - times, N_1, N_2, p_j = p|\sum U_j$, we have to choose indexes x and y, such that $x + y \to \max$ and jobs $j_x, j_{x+1}, \ldots, j_{n_1}, i_y, i_{y+1}, \ldots, i_{n_2}$ can all be processed on-time at the beginning of a schedule. Thus, we have to take into account at most $(n_1{+}1)\log(n_2{+}1)$ pairs (x, y). For each of the pairs we solve the problem $1|setup{-}times, N_1, N_2, p_j = p|\sum T_j$ with set of jobs $\{j_x, j_{x+1}, \ldots, j_{n_1}, i_y, i_{y+1}, \ldots, i_{n_2}\}$ by a modification of Algorithm 1. If $\sum T_j(\pi^*) = 0$, then pair (x, y) is feasible. We can conclude the following (see Lemma 8).

Lemma 8 *The problem* $1|setup{-}times, N_1, N_2, p_j = p|\sum U_j$ *can be solved in* $O(n^7 \log n)$ *time.*

For the problem $1|setup - times, N_1, N_2, p_j = p|\sum w_j U_j$, a dynamic programming polynomial time algorithm is suggested. This algorithm based on the following assumptions. Note jobs in $N = \{H_1, H_2, \ldots, H_n\}$, where $w_{H_1} \leq w_{H_2} \leq \ldots \leq w_{H_n}$. If $w_{H_k} = w_{H_{k+1}}$, then $d_{H_k} \leq d_{H_{k+1}}$. Jobs from N_1 and N_2 are noted and ordered according to Lemma 6. Let $H_n \in N_2$ and $H_n = i_k$. For H_n a position in a schedule is defined by a pair (t, l), where $t \in \Theta$ is the starting time of the job, the index $l = 0, 1, \ldots, n_1$ means that on-time jobs from the subset $\{j_1, j_2, \ldots, j_l\}$ precede the job H_n in a schedule and on-time jobs from the subset $\{j_{l+1}, j_{l+2}, \ldots, j_{n_1}\}$ are scheduled after H_n. A position $(-, n_1 + 1)$ means that the job H_n is late and processed at the end of schedule from time $T \in \Theta$.

Then, for each position (t, l) among $O(n^4)$ possible, we can decompose the initial problem into two independent subproblems:

- with a set of jobs $N_{left} = \{j_1, j_2, \ldots, j_l, i_1, i_2, \ldots, i_{k-1}\}$ which have to be processed in interval $[0, t)$;
- with a set of jobs $N_{right} = \{j_{l+1}, j_{l+2}, \ldots, j_{n_1}, i_{k+1}, i_{k+2}, \ldots, i_{n_2}\}$ which have to be processed in interval $[t + p, T)$;

The running time of the Algorithm is $O(n^{15})$.

5 Conclusion

We suppose that running times of Algorithms can be substantivally reduced after their more detailed analysis. Another question arises as for single machine equal-processing-time scheduling problems without setup-times and precedence relations: "Are there problems with equal processing time of jobs, which are NP-hard?"

Acknowledgements The work was partially supported by RFBR (Russian Foundation for Basic Research): 11-08-01321, 11-08-13121, by CNRS (contract 363745) and Saint-Etienne Métropole, France.

References

1. S.Coeney, F.C.R. Spieksmay and G. V. Berghe (2011), The Lockmaster's Problem, ATMOS 2011, Saarbruecken.
2. R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1979). Optimization and approximation in deterministic machine scheduling: a survey. Ann. Discrete Math., 5, 287 –326.
3. S. Kravchenko and F. Werner (2011). Parallel Machine Problems with Equal Processing Times: A Survey. Journal of Scheduling, 14(5), 435 - 444.
4. E.S. de Oliveira (2001). Solving Single Track Railway Scheduling Problem Using Constraint Programming. The University of Leeds. School of Computing. PhD Thesis.

MISTA 2013

An Iterated Local Search Algorithm for the Technician Routing and Scheduling Problems

Nasser R. Sabar • Mohd Zakree Ahmad Nazri

1. Introduction

Among many problems in the transportation and logistics filed, vehicle routing problem (VRP) is the most widely studied problem by the researcher community [1]. In VRP, the main goal is to design a set of vehicle routes to serve a set of customers with known demand at the minimum cost [2]. Cost reduction is the main factor in solving VRP, which usually represents the total traveling distance or the total time, needed to serve all customers [2]. Over the years, several variants of the VRP have been introduced such as the capacitated vehicle routing (CVRP) and the vehicle routing problem with time windows (VRPTW) [3]. One of the recent variant of the VRP is the technician routing and scheduling problem (TRSP). TRSP is introduced by Pillac et al. [4] as an extension of the VRPTW with additional constraints. TRSP can be defined as follows: given a set of technicians each one has a set of skills, tools and spare parts, and a set of requests that need specific skills, tools and/or spare parts, design a set of tour to sever the required requests at the minimum cost (i.e., traveling distance). Each request should be visited only once, served by a technician with required skills, tools and spare parts for this request and during its time window [4]. TRSP has many real world applications such as public utilities, telecom services, maintenance operation and home care scheduling problems. As TRSP is an extension of the VRPTW, TRSP is also NP-Hard problems [4]. Thus, exact methods are only advisable for small size problems. Consequently, heuristic and meta-heuristic approaches are desirable to solve VRPTW and TRSP, as they retune a reasonably good solution within acceptable computation times [2, 3, 4].

In this work, we propose an iterated local search algorithm (ILS) for the TRSP. ILS is an iterative single solution based method that improves the given solution through three phases known as perturbation, local search and acceptance criterion. The proposed ILS is tested on a set of TRSP instances introduced by [4]. Results demonstrate that the proposed ILS is very competitive, if not superior, compared to the best methods in the literature.

2. The proposed method

In this section, we first describe the basic iterated local search algorithm (ILS) followed by our proposed ILS for the TRSP.

N.R. Sabar • M. Z. A. Nazri
Data Mining and Optimization Research Group (DMO), Centre for Artificial Intelligent Technology (CAIT), Universiti Kebangsaan Malaysia, 43600 UKM Bangi Selangor, Malaysia.
E-mail: naserdolayme@yahoo.com. mzan@ftsm.ukm.my.

N. R. Sabar
The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih Selangor, Malaysia.

2.1 Basic iterated local search algorithm

Iterated local search algorithm (ILS) is an iterative single solution improvement based method [5]. The main idea of ILS is that the local search algorithm can be improved by generating a new starting solution via perturbing a visited local optima solution [5]. In other words, the performance of the local search algorithm can be improved via restarting the search by perturbing the local optima solution instead of generating completely a new solution. ILS has three phases known as local search phase, perturbation phase and acceptance criterion phase. Algorithm 1 shows the pseudo code of a basic ILS [5].

Algorithm 1: Basic Iterated Local Search (ILS)	
1	*Set MaxIter; Iter ← 0;*
2	S_0 ← GenerateInitialSolution();
3	S_1 ← Local search (S_0);
4	**while** (Iter < MaxIter) **do**
5	S_2 ← Perturbation phase(S_1);
6	S_3 ← Local search phase (S_2);
7	S_1 ← Acceptation criteria phase (S_1, S_3, *history*);
8	*Iter=Iter+1;*
9	end **while**
10	Return the best solution

First, ILS set the maximum number of iterations (line 1), generates an initial solution (line 2) and applies a local search algorithm to improve the generated initial solution (line 3). Then, ILS executes the while loop (lines 4-9) for a predefined number of iterations. At each iteration, a perturbation phase (line 5) is invoked to generate a new starting solution (S_2) for the local search phase by perturbing the local optimum solution (S_1). Next, the local search phase is iteratively applied to improve a given solution (S_2) (line 6). Then, the generated solution by the local search phase (S_3) is either accepted or rejected based on the utilized the acceptance criterion (line 7). Next, update the iteration counter (*Iter*) (line 8) and start a new iteration.

2.2 The proposed ILS for TRSP

In this section, we describe the main components of the proposed ILS which are: the initial solution generation method, the perturbation phase, local search phase and the acceptance criterion phase.

1- **Initial solution**: in this work, the initial solution is generated by a regret heuristic [4][6]. Regret heuristic is a greedy heuristic that integrates a look-ahead feature. At each iteration, a request with the highest *regret* value is inserted into the best possible place, where the *regret* value represents the additional cost that will be added if a request is not inserted at the best place [4][6].

2- **Perturbation phase**: The main role of the perturbation phase is to generate a new starting solution by perturbing (modifying) a given local optima solution. Usually, perturbation size is very crucial, as it determine the success of the ILS. A small perturbation size may retune the search into the same point; whilst a large perturbation size makes ILS inefficient, as ILS will be like a random multi-start method. In this work, we use three different perturbation operators (moves) to perturb (modify) a given local optima solution. The utilized perturbation operators are inspired from the adaptive large neighborhood search idea [4][7]. At each iteration of

the perturbation phase, a remove operator will be applied to remove a number of requests from the current solution followed by a repair operator to re-insert the removed requests. In this work, a remove operator is randomly selected to remove a set of requests from a given solution. The utilized remove operators are:

i- *Random*: randomly remove a set of requests from the current solution.
ii- *Critical*: remove the most costly set of request from the current solution.
iii- *Related*: first randomly select a set of request and remove all the related requests to the selected ones.

The removed requests are then re-inserted into the current solution by applying the following repair operators in consecutive manner:

i- Sort the removed request based on their *regret* value and re-insert them into the best possible position.
ii- Select a removed request from a given set and re-inset it into a random position.
iii- Select a removed request with the smallest *regret* value and re-insert it into a random position.

In this work, repair operators are repeatedly applied until all removed requests have been re-inserted into the current solution.

3- Local search phase: In this work, we employ the variable neighborhood decent (VND) local search algorithm [8]. VND starts with an initial solution obtained from the perturbation phase and iteratively explore its neighborhood solutions using a set of neighborhood operators for a predefined number of iterations. VND works with K_{max} neighborhood operators N_k, k=1… K_{max}, in a predefined sequence. VND starts with an initial solution (S) and set k=1. If the quality of the generated neighborhood solution (S`) by $N_k$ is better than $S$, then replace $S$ with $S$` and set k=1. Otherwise, increase k by one (k=k+1) and apply the next neighborhood operator. VND will stop the searching if the local optima of the N_{Kmax} cannot be improved for a pre-defined number of iterations. In this work, the utilized neighborhood operators are: 2-opt*, 2-opt, random move and random swap. Generally, the sequence of the neighborhood operators plays an important role in the performance of VND. This is because different neighborhood operator generates different landscape structure and the best sequence is usually dependent on the current state of the search as well as the problem instance. In this work, the sequence of the neighborhood operators for the VND is determined by recording the previous performance of each neighborhood operator and then sorting the given neighborhood operators based on their performance measure from the highest to the lowest. The performance of the i-th neighborhood operator is calculated as follows:

$$m(i) = \sum_{j=1}^{n_s} n_i^j / n_s \quad (1)$$

$$n_i = \begin{cases} (f_1 - f_2)/|f_1| & \text{if the operator improved the solution at } i \text{ iteration} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Where n_s represents number of time that the *i*-th neighborhood operator has been applied at the given solution up to current iterations. f_1 is the given solution and f_2 is the generated solution by the *i*-th neighborhood operator. Initially, the neighborhood operators sequence is randomly generated. Then, at each subsequent call of the local search phase, VND will sort the given set of the neighborhood operators based on their previous performance measure from the highest to the lowest one.

4- **Acceptance criterion**: In this work, we use exponential Monte Carlo acceptance criterion [9]. Improving solutions are always accepted. Worse solutions are accepted with a probability of $R < \exp(-\delta)$, where R is a random number between [0, 1] and δ is the change in the solution quality. The probability of accepting worse solutions will decrease as δ increases [9].

3. Computational results

The performance of the proposed ILS is validated on the 56 TRSP instances introduced by [4]. These instances are generated based on Solomon VRPTW benchmarks by adding skills, tools, and spare parts information to each of the VRPTW instances. The generated TRSP instances have 25, 50 or 100 requests, either randomly distributed in a 100 X 100 square or in clusters [4]. The parameter settings of the proposed ILS are listed in Table 1. These parameter values are fixed based on a preliminary test.

Table 1 The parameter settings of the ILS

#	Parameter	Value
1-	MaxIter of ILS	1000
2-	Max non-improvement iteration for the VND	20
3-	Number of request to be removed	Randomly between [0.1*n*, 0.4*n*], where *n* is the number of requests in a given instance

Table 2 presents the average gap to the best known results of ILS over 31 runs for the six groups of TRSP instances along with the average results of pALNS [4]. Best results are highlighted in bold. The values reported in Table 2 represents the total traveling distance, i.e., the smaller the better. The first column represents the instance group, second column reports our results, third column shows pALNS results and last two columns show the average time in second for ILS and pALNS. As can be seen from Table 2, the proposed ILS produced better average results for 4 out of 6 groups of instances compared to pALNS. Of 4 out of 6 groups of instances, the computational time of ILS is lower than pALNS. In addition, the overall average of the average results and the computational time of ILS are lower than pALNS (see last row in Table 2). Generally, the favorable results achieved by ILS suggest that ILS is very competitive when compared to pALNS over all TRSP instances. Both our proposed ILS and ALNS use a similar idea of remove and repair to modify the current solution. However, the proposed ILS further improves the perturbed solution by using a local search phase (VND in this work). In addition, the proposed ILS deals with different instance landscape structures by generating, during each call of the local search phase (VND), a different sequence of neighborhood operators.

Table 2 The computational results of ILS compared to pALNS

Group	Our results (ILS)	pALNS	ILS Time (s)	pALNS Time (s)
C1	**1**	1.22	**15**	24.0
C2	**0.66**	0.78	**20**	27.8
R1	**3.11**	4.96	**21**	28.9
R2	1.71	**1.69**	**24**	31.0
RC1	**3.62**	3.90	28	**27.9**
RC2	2.14	**1.93**	34	**27.9**
Overall	**2.04**	2.54	**23.66**	28.1

4. Conclusion

This work has presented an iterated local search algorithm for the technician routing and scheduling problems. Iterated local search algorithm starts with an initial solution and iteratively explores its neighborhood solution by successively applying a perturbation phase and local search phase. In this work, at the perturbation phase, we have applied three different perturbation operators to modify the current local optima solution. While, at the local search phase, the variable neighborhood decent is used to further improve a perturbed solution. The proposed algorithm is applied on the available TRSP benchmark instances and obtained very good results when compared to the best method in the literature.

References

1. G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management science*, vol. 6, pp. 80-91, 1959.
2. Savelsbergh, M.: The vehicle routing problem with time windows: minimizing route duration. INFORMS J. Comput. 4(2), 146–154 (1992).
3. Solomon, M. M. (1987). Algorithms for the vehicle-routing and scheduling problems with time window constraints. Operations Research, 35(2):254-265.
4. Pillac, V., Guéret, C.,Medaglia, A.L.: A parallel matheuristic for the technician routing and scheduling problem: supplementary material (online) (2011). http://hdl.handle.net/1992/1145.
5. Lourenco, H.R., Martin, O., Stutzle, T.: Iterated local search: Framework and applications. In: ¨Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 146, chap. 9, pp. 363–397. Springer, New York, NY, 2 edn. (2010).
6. Potvin, J.Y., Rousseau, J.M.: A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. Eur. J. Operat. Res. **66**(3), 331–340 (1993).
7. Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. Computers & Operations Research, 34(8):2403-2435.
8. Hansen, P., Mladenovi´c, N., Moreno Pérez, J.: Variable neighbourhood search: methods and applications. 4OR 6, 319–360 (2008).
9. Ayob, M. and Kendall, G. (2003). "A Monte Carlo Hyper-Heuristic To Optimise Component Placement Sequencing For Multi Head Placement Machine". (pdf) Proc. of the International Conference on Intelligent Technologies, InTech'03, Chiang Mai, Thailand, 17-19 Dec, 132-141.

MISTA 2013

The Dynamic Patient Admission Scheduling with Operating Room Constraints, Flexible Horizon, and Patient Delays

Sara Ceschia · Andrea Schaerf

1 Introduction

We propose an extension and a revision of the Patient Admission Scheduling (PAS) problem defined in a previous work [5]. The revision consists in considering the dynamic version of the problem, which includes the stochastic arrival and discharge of patients (including urgent ones), along the lines proposed in [3, 8]. The main extension is that we integrate constraints about the utilisation of operating rooms for patients that have to undergo a surgery, revising our previous work [4]. Considering also the operating theatre in the scheduling process modifies the problem significantly and requires the definition of a more elaborate model. This new model has to include the complex management of delay of the patient admission, a flexible planning horizon, and new components of the objective function which take into account also the utilisation of hospital resources.

2 Main new features

With respect to the basic PAS problem [5] and to other previous proposals[1], the distinguishing features of our work are the following ones.

– Possibility to delay patients also beyond the planning horizon. This extension allows us to capture the cases, that frequently occur in practice, in which the nominal occupancy is above 100%, meaning that the overall capacity of the hospital is not sufficient to treat all the patients.
In this way, some of the patients have to be postponed to a new planning phase. The selection of the patients to be delayed need obviously to be based on their urgency, but also the overall objectives of the hospital must be considered (efficiency, cost reduction, . . .).

S. Ceschia and A. Schaerf
DIEGM, University of Udine
Via delle Scienze 206
E-mail: {sara.ceschia, schaerf}@uniud.it

- Due to the extensive use of delays, we also need to refine the component of the objective function that accounts for their costs. Indeed, delays cannot be treated in a homogeneous way, but their penalty is related to the discomfort caused by long and repeated delays of the same patient.
- In addition, to provide against the excessive resort to the option of delaying patients, we have to include a new cost component that accounts for the under-use of resources (mainly operating rooms, but also beds). This new component is conceived so that it penalises more the under-use in the current planning day and the near ones, whereas it has a limited impact for the distant days, given that it could be balanced by new-coming unexpected patients.
- In order to make the formulation fully realistic, it is necessary also to take into account the fact that a hospital is never completely empty but it runs continuously; therefore the initial situation has to include patients already present, which should be scheduled for the subsequent days, but considering their initial assignment to a room.
- As customary, operating rooms are managed according to the so-called master surgical schedule (MSS) which is a cyclic timetable that defines, for each day of the cycle, the set of operating room time blocks assigned to a team of surgeons [7]. In general, however, the strict structure prescribed by the MSS might need to be relaxed, so as to enable a better scheduling. In this work, we permit surgeries to be scheduled in an operating room time block even if their total duration exceed the length of the timeslot assigned to the corresponding specialty. This might result in the possibility that the operating room stays open for some limited extra-time, which is properly penalised by an ad-hoc component of the objective function.

3 Instance generator

Mainly due to privacy concerns, it is in general very difficult to obtain real data in the healthcare domain. The PAS problem is no exception, and unfortunately, there are no real-world instances available at present for this problem. It is therefore very important to develop an instance generator that could produce very realistic datasets.

To this aim, we have developed a generator that uses available real world data (e.g., standard duration of surgery and length of stay of patient for each specialty) and known distributions about patient behaviour. Using these data, the generator is able to synthesise realistic and challenging case studies, parameterized for all possible sizes and configurations of the hospital.

The generator and the relevant generated instances will be made available on the web, in XML format, along with our solutions and a solution validator.

4 Search method and experimental results

We have designed a solution method based on Simulated Annealing, which explores the search space using complex neighbourhood operators. This approach is based on our previous solver [2] that obtained the best known results for the public instances of the PAS problem, some of which (5 out of 13) have been improved recently by a different technique [6].

This work is still in progress, and we are performing an extensive experimental analysis of our solution method. The analysis considers a statistically-principled tuning phase that involves the main parameters of Simulated Annealing, namely the starting temperature and the cooling scheme. It also includes a sensitivity analysis for the cost components of the model and their corresponding weights in the objective function.

Finally, we are performing a comparison between the static version of the problem, which considers all patients as elective, and the dynamic one, which takes into account also urgent patients. This investigation could highlight what is the importance of having or not the information about fore-coming patients enough in advance, and which techniques can be devised to manage uncertainty and achieve good performance using on-line scheduling approaches.

References

1. Burak Bilgin, Peter Demeester, Mustafa Misir, Wim Vancroonenburg, and Greet Vanden Berghe. One hyper-heuristic approach to two timetabling problems in health care. *Journal of Heuristics*, 18(3):401–434, 2012.
2. Sara Ceschia and Andrea Schaerf. Local search and lower bounds for the patient admission scheduling problem. *Computers & Operations Research*, 38(10):1452–1463, 2011.
3. Sara Ceschia and Andrea Schaerf. Modeling and solving the dynamic patient admission scheduling problem under uncertainty. *Artificial Intelligence in Medicine*, 56(3):199–205, 2012.
4. Sara Ceschia and Andrea Schaerf. Patient admission scheduling with operating room constraints. In *Proc. of the 9th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2012)*, 2012.
5. Peter Demeester, Wouter Souffriau, Patrick De Causmaecker, and Greet Vanden Berghe. A hybrid tabu search algorithm for automatically assigning patients to beds. *Artificial Intelligence in Medicine*, 48(1):61–70, January 2010.
6. T.M. Range, R.M. Lusby, and J. Larsen. A column generation approach for solving the patient admission scheduling problem. Technical report, Department of Business and Economics, University of Southern Denmark, 2013.
7. J. M. van Oostrum, M. Van Houdenhoven, J. L. Hurink, E.W. Hans, Wullink G., and Kazemier G. A master surgical scheduling approach for cyclic scheduling in operating room departments. *OR Spectrum*, 30(2):355–374, 2008.
8. Wim Vancroonenburg, Patrick De Causmaecker, and Greet Vanden Berghe. Patient-to-room assignment planning in a dynamic context. In *Proc. of the 9th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT-2012)*, 2012.

MISTA 2013

Scheduling teams for aircraft ground maintenance

Giuseppe Lancia · Paolo Serafini

1 Introduction

This talk deals with a real application concerning the scheduling of the maintenance staff of an airport and the described procedures are currently being implemented in a large Italian airport.

The time between the arrival and the departure of an aircraft is usually devoted to some maintenance operations carried out by the airport staff. The operations are requested by the aircraft crew just before the arrival and they are not known in advance. The operations require different skills and hence the airport staff must have enough workers available with the proper skills to cope with the requests.

The airport staff is organized according to a duty table. On each day each worker may work in the morning shift (M) or in the afternoon shift (A) or in the night shift (N) or he may rest (R). The duty table has rows corresponding to 10 weeks and columns corresponding to the seven days of the week. Each entry in the table is one of the four symbols (M,A,N,R). The table is fixed and cannot be changed. On a given week, each worker has to follow a particular row of the duty table, then he moves to the next row in the next week and so on in a cyclic way, i.e., going to row 1 after completing the last row. The only difference between workers is their starting rows in the duty table.

The decision variables are the starting rows in the duty table for each worker. Clearly deciding the starting rows greatly affects the availability of workers in a particular shift. The time table of arrivals and departures is known much in advance, and this enables to consider a time horizon for scheduling the staff of six months, but the actual requests are known only at the last minute. Therefore the possible requests must be estimated on a statistical basis.

Giuseppe Lancia
University of Udine
E-mail: giuseppe.lancia@uniud.it

Paolo Serafini
University of Udine
E-mail: paolo.serafini@uniud.it

We have split the problem into the following three steps: evaluation of the requests; evaluation of the work force during a shift; decision of the best starting rows.

2 Evaluation of the requests

Each maintenance task requires workers who possess a particular skill and a specific level for that skill. For each skill there are three levels: level 0 (the lowest), corresponds to no particular competence and is possessed by every worker; level 1 (medium), corresponds to a good specialization in the specific skill and level 2 (the highest), corresponds to very advanced competence.

For each skill, the request coming from the aircraft crew is one of the following

$$\{\}, \quad \{0\}, \quad \{1\}, \quad \{2\}, \quad \{01\}, \quad \{02\}, \quad \{12\}, \quad \{11\}$$

denoting respectively no request at all (for that skill), one worker of level 0 (hence not associated to any particular skill), one worker of level 1, one worker of level 2, a pair of workers with levels 0 and 2 etc. For each flight and each skill the probabilities of the previous eight outcomes are known. From these probabilities it is not difficult (via generating functions which are just polynomials in this case [2]) to compute the distribution functions

$$d^k_{hr} = \Pr\left\{Y^k_h \leq r\right\}$$

where Y^k_h is a random variable which counts how many workers with skill k and level h will be requested during a particular shift. The values d^k_{hr} assign the confidence level such that Y^k_h will have value not larger than r. These values form a discrete set but we interpolate them linearly in order to have a continuous distribution. Therefore at the end of this phase we have available, for each skill and each level, a *real* value r such that $d^k_{hr} = 0.95$ (having assigned a confidence level of 95%).

3 Evaluation of the work force during a shift

Given a certain set of available workers (each one with his own skill and level) in a specific shift, we want to evaluate if the set meets in a satisfactory way the maintenance requests for that shifts. The requests are summarized by the previous r values that we may label as r_j, with j ranging on the pairs skill-level, plus an entry for the 0 level. We may model the problem with a bipartite graph $G = (W, R, E)$, where W is the set of available workers in the shift and R is the set of requests. The edge set E consists of the pairs (i, j) where the worker i has the skill-level j. To each $j \in R$ a real value r_j is associated and to each $i \in W$ a real value w is associated (the same for all $i \in W$). The value w has to be regarded as the working capacity of a single worker. Normally it should be $w = 1$. However, it is safe to take into account possible absences from work for various reasons and reduce the value to $w = 0.7$.

Although the problem deals with integer quantities, like workers and tasks, nonetheless, due to the uncertainty connected to the task requirement and also to the worker presence, we may disregard the integrality constraints and think of fractional assignments of workers to tasks. By-and-large these tasks will not be requested all at the same time and it is conceivable that a worker may devote part of his time to one task and another part to some other task.

Hence we want to find an assignment x_{ij}, $(i,j) \in E$, such that

$$\sum_{j \in R} x_{ij} \leq w, \qquad i \in W$$

and some measure $\delta(x)$ of deviation from the values r_j, $j \in R$, of

$$x(j) := \sum_{i \in W} x_{ij}$$

is as small as possible. We have chosen the following measure

$$\delta(x) := \max_{j \in R} \max \left\{ r_j - x(j) \; ; \; \alpha \left(x(j) - r_j \right) \right\}$$

where the surplus is weighted by α. We find the assignment minimizing $\delta(x)$ by carrying out a double binary search solving each time a max flow problem ([1]). The optimal value δ^* is a performance measure of the workers team allocated to a particular shift.

The total number of shifts in the duty table is rather high and therefore we need a fast algorithm to evaluate all shifts for a particular set of starting rows of the workers.

4 Deciding the workers' starting rows in the duty table

Now we want to find the best starting rows for all workers so that in each shift the possible requests can be met. It is not immediate to understand how moving the starting row of a single worker from one row to another one affects the whole schedule. Hence we may opt for a local search procedure in which a simple move, i.e., moving a single worker to another row, is performed randomly and this new schedule is immediately evaluated according to the previous steps.

5 Conclusions

In the actual implementation the management has available a set of parameters to interact with the procedure and to 'drive' the solution toward a good value. The essential point is that the core routines must be fast and robust to produce a reliable schedule. The management seems satisfied with the software.

References

1. R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network flows. Theory, algorithms and applications*, Prentice Hall, New Jersey, 1993.
2. Graham, R.L., D.E. Knuth, O. Patashnik, *Concrete mathematics: a foundation for computer science*, 2 ed., Addison Wesley, Reading, MA, USA, 1994.

MISTA 2013

Extending the late acceptance metaheuristic for multi-objective optimization

Wim Vancroonenburg · Tony Wauters

1 Introduction

The present paper proposes an extension to the late acceptance metaheuristic introduced by (Burke and Bykov, 2008, 2012) so that it can solve multi-objective optimization problems. Late acceptance was successfully applied to several single-objective optimization problems such as examination timetabling problems (Burke and Bykov, 2008; Ozcan et al, 2009), the lock scheduling problem (Verstichel and Vanden Berghe, 2009), course timetabling problems (Abuhamdah, 2010), and the machine reassignment problem (Vancroonenburg and Wauters, 2012; Gogos et al, 2012).

Liefooghe et al (2012) state that, in comparison to the tremendous number of existing multi-objective evolutionary algorithms, the number of multi-objective local search approaches is extremely small. This paper proposes such a procedure, which we call multi-objective late acceptance (MOLA).

2 Multi-objective late acceptance

The basic idea behind Late Acceptance (LA) is to compare a new solution with the solution generated L iterations ago. This strategy introduces diversification in the search process, allowing the search algorithm to move freely in the solution space to some extent, while still pushing the search towards improving solutions. The main parameter L of the algorithm determines how strong the diversification is: higher values of L result in more diversification. Burke and Bykov (2012) improved the idea of LA to the Late Acceptance Hill Climbing (LAHC) variant by also checking the new candidate solution against the current solution.

Algorithm 1 (denoted MOLA) describes how LAHC can be used in a multi-objective setting. The main idea of LAHC is preserved in MOLA, namely comparing with the solution obtained L iterations ago. This is typically implemented by maintaining a cyclic list (denoted Q) of length L, containing the objective function value of the last

Wim Vancroonenburg, Tony Wauters
CODeS, KU Leuven KULAK
CODeS, KAHO Sint-Lieven
E-mail: wim.vancroonenburg, tony.wauters@kahosl.be

L current solutions. In the multi-objective setting, the list stores the objective vectors $\mathbf{f}(S)$ of the last L current solutions.

The algorithm starts from an initial solution S_0, and the late acceptance list Q is initialized with the objective vector $\mathbf{f}(S_0)$. Each iteration, a new candidate solution S' is generated from the neighbourhood $N(S)$ of the current solution. The new candidate solution is then compared and possibly accepted as the new current solution according to the Pareto dominance relation:

Definition 1 Pareto dominance (?). An objective vector $u = (u_1, \ldots, u_n)$ is said to dominate $v = (v_1, \ldots, v_n)$ (denoted by $u \prec v$) if and only if no component of v is smaller than the corresponding component of u and at least one component of u is strictly smaller, that is; $\forall i \in \{1, \ldots, n\} : u_i \leq v_i \wedge \exists i \in \{1, \ldots, n\} : u_i < v_i$

Furthermore, the algorithm maintains a set of 'best' solutions (denoted P) that are pairwise non-dominating. Each time a new candidate is accepted, the algorithm checks whether or not the candidate dominates any solutions in the pareto set, and if so, the candidate is added to the set. The set is also updated to remove solutions that have now become dominated. Finally, at the end of each iteration the late acceptance list Q is updated to include the objective vector $\mathbf{f}(S)$ of the (new) current solution.

Algorithm 1 The multi-objective late acceptance algorithm

Require: L, S_0
  $P \leftarrow \{S_0\}$
  $S \leftarrow S_0$
  for $i \leftarrow 1 \ldots L$ **do**
    $Q_i \leftarrow \mathbf{f}(S_0)$
  end for
  $i \leftarrow 0$
  while stopping condition not met **do**
    $S' \leftarrow N(S)$
    if $\mathbf{f}(S') \prec \mathbf{f}(S)$ or $\mathbf{f}(S') \prec Q_{[i \bmod L]}$ **then**
      $S \leftarrow S'$
      $UPDATE_PARETO_SET$(P,S)
    end if
    $Q_{[i \bmod L]} \leftarrow \mathbf{f}(S)$
    $i \leftarrow i + 1$
  end while
  return P

Figure 1 illustrates the basic principle of MOLA. If the candidate solution would only have been compared with the current solution, then it would not have been accepted. However, it is compared to the solution obtained L iterations ago and therefore it is accepted and considered for inclusion in the pareto set.

3 Experimental Setup

In order to asses the performance of MOLA, several computational experiments on well known multi-objective benchmarks will be presented at the conference, including multi-objective versions of the permutation flowshop scheduling problem, the traveling salesman problem and the quadratic assignment problem. MOLA will be compared

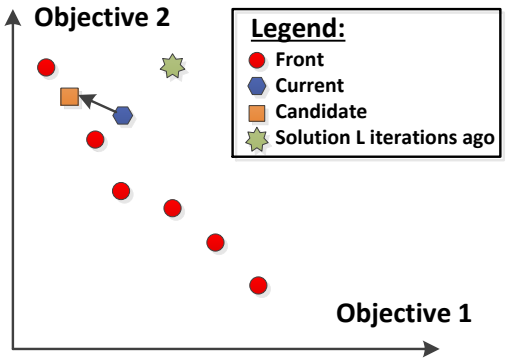


Fig. 1 A two-objective minimization example illustrating the MOLA acceptance strategy. Comparing against the current solution would result in not accepting the candidate solution. However, the candidate solution dominates the solution obtained L iterations ago, thus it is accepted as the new current solution.

with other multi-objective local search methods, such as multi-objective simulated annealing (Smith et al, 2004; Bandyopadhyay et al, 2008) and different variants of pareto local search (Dubois-Lacoste et al, 2011; Drugan and Thierens, 2012). Preliminary results are very promising.

References

Abuhamdah A (2010) Experimental result of late acceptance randomized descent algorithm for solving course timetabling problems. IJCSNS International Journal of Computer Science and Network Security 10(1):192–200

Bandyopadhyay S, Saha S, Maulik U, Deb K (2008) A simulated annealing-based multiobjective optimization algorithm: Amosa. IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION 12(3):269–283

Burke EK, Bykov Y (2008) A late acceptance strategy in hill-climbing for exam timetabling problems. In: Proccedings of PATAT 2008 conference, Montreal, Canada

Burke EK, Bykov Y (2012) The late acceptance hill-climbing heuristic, technical report csm-192. Tech. rep., Computing Science and Mathematics, University of Stirling

Drugan MM, Thierens D (2012) Stochastic Pareto local search: Pareto neighbourhood exploration and perturbation strategies. Journal of Heuristics 18(5):727–766

Dubois-Lacoste J, López-Ibáñez M, Stützle T (2011) A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. Computers & Operations Research 38(8):1219–1236

Gogos C, Valouxis C, Alefragis P, Housos E (2012) Attacking googles machine reassignment problem using mathematical programming, metaheuristics and heuristics. In: Proceedings of European Conference on Operational Research (EURO2012), Vilnius, Lithuania

Liefooghe A, Humeau J, Mesmoudi S, Jourdan L, Talbi EG (2012) On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. Journal of Heuristics 18:317–352

Ozcan E, Bykov Y, Birben M, Burke EK (2009) Examination timetabling using late acceptance hyper-heuristics. In: Proccedings of the 11th conference on Congress of Evolutionary Computation, Trondheim, Norway

Pareto V (1896) Cours d'conomie politique. Rouge, Lousanne, Switzerland

Smith K, Everson R, Fieldsend J (2004) Dominance measures for multi-objective simulated annealing. In: Evolutionary Computation, 2004. CEC2004. Congress on, vol 1, pp 23 – 30 Vol.1

Vancroonenburg W, Wauters T (2012) A late acceptance metaheuristic for the machine reassignment problem. In: Proceedings of European Conference on Operational Research (EURO2012), Vilnius, Lithuania

Verstichel J, Vanden Berghe G (2009) A late acceptance algorithm for the lock scheduling problem. Logistic Management 5:457–478

MISTA 2013

Exact algorithms for single-machine scheduling with time windows and precedence constraints

Morteza Davari · Erik Demeulemeester · Roel Leus · Fabrice Talla Nobibon

1 Introduction

This paper presents exact algorithms for solving a total weighted tardiness scheduling problem on a single machine, where each job is characterized by a processing time, a release date, a due date, a deadline and a weight representing the penalty per unit-time delay in the completion of that job. The execution of a job cannot start before its release date and must be completed no later than its deadline. There are precedence constraints among jobs and a job can be executed only when all its predecessors are completed. The goal is to minimize total weighted tardiness penalties while respecting scheduling constraints.

During the last 50 years, many papers have been published dealing with various single-machine scheduling problems for different objective functions including the total weighted tardiness [9]. Scheduling with release dates has been studied by several authors [1,2,6,10–12]. Held and Karp [2] present a dynamic programming (DP) approach for $1||\sum w_jT_j$. Potts and van Wassenhove [10] propose a branch-and-bound (B&B) algorithm to solve $1||\sum w_jT_j$. Akturk and Ozdemir [1] and Jouglet et al. [4] develop B&B algorithms which can solve small and medium-sized instances of $1|r_j|\sum w_jT_j$. Tanaka and Fujikuma [12] propose a successive sublimation dynamic programming (SSDP) approach that is able to solve instances of $1|r_j|\sum w_jT_j$ with up to 100 jobs. Tanaka and Sato [13] propose an SSDP algorithm that optimally solves instances of $1|prec|\sum g_j(C_j)$ where g_j is a piece-wise linear function of the completion time of job j.

To the best of our knowledge the general problem of single-machine scheduling with release dates, deadlines and precedence constraints has not yet been studied in the literature.

Department of Decision Sciences and Information Management
Faculty of Business and Economics, KU Leuven
E-mail: <firstname>.<lastname>@kuleuven.be

2 Problem statement

We consider a set $N = \{1, 2, ..., n\}$ of jobs. Job i is characterized by a processing time p_i, a release date r_i, a due date d_i, a deadline δ_i and a weight w_i that represents the cost per unit time of delay beyond d_i. Jobs can neither be processed before their release dates nor after their deadlines ($0 \leq r_i \leq d_i \leq \delta_i$). We are also given a graph $G = (N, A)$ of precedence constraints. The arc $(i, j) \in A$ means that job i must be executed before job j (job i is a predecessor of job j). A job can only be scheduled after the completion times of all its predecessors.

The decision to be made is: in which order to process the jobs so that the total weighted tardiness is minimized. We denote by π an arbitrary sequence of jobs where π_k represents the job at the k^{th} position in that sequence. The weighted tardiness associated with each job in the sequence is given by $\theta_i(\pi) = w_{\pi_i} (C_{\pi_i} - d_{\pi_i})^+$ where $x^+ = \max\{0, x\}$ and C_{π_i} is the completion time of the job at the i^{th} position in the sequence π and is computed by

$$C_{\pi_i} = \begin{cases} \max\{r_{\pi_i}, C_{\pi_{i-1}}\} + p_{\pi_i} & \text{if } i > 1 \\ r_{\pi_i} + p_{\pi_i} & \text{if } i = 1 \end{cases} \tag{1}$$

We denote by $\mathcal{D}$ the set of all feasible permutations. In other words, a permutation π is feasible ($\pi \in \mathcal{D}$) if and only if

$$r_{\pi_i} + p_{\pi_i} \leq C_{\pi_i} \leq \delta_{\pi_i} \text{ where } i = 1, 2, ..., n \tag{2}$$

$$\text{for each pair of jobs } (\pi_i, \pi_j) \in A \text{ we have } i < j. \tag{3}$$

Note that it is possible that $\mathcal{D} = \emptyset$. An intuitive formulation for our General Scheduling Problem (GSP) is as follows:

$$\text{GSP}: \min_{\pi \in \mathcal{D}} g(\pi) = \sum_{i=1}^{n} \theta_i(\pi) \tag{4}$$

This problem is at least as hard as the single-machine total weighted tardiness problem with release dates, which is known to be strongly NP-hard [6, 9].

3 Solution approaches

In this work, we study a number of exact solution approaches for GSP. The intuitive formulation for the GSP presented above is not linear, therefore it cannot be solved by a standard Mixed Integer Programming (MIP) solver. We propose an Assignment Formulation (ASF) and a Time-Indexed Formulation (TIF) for the problem. These formulations are modifications of those presented in [5, 11]. The SSDP algorithm presented in [13] solves those instances of GSP where an optimal solution without machine idle time exists; this is the case in particular when $r_j = 0$ and $\delta_j = \infty$. To be able to solve all instances of GSP, one can add dummy jobs to eliminate the possibility of machine idle time, as described in [12].

We also present a B&B algorithm that optimally solves GSP. This algorithm is inspired by the B&B algorithm proposed by Jouglet et al. [4]. We modify existing lower bounds and dominance rules [3, 4, 7, 8, 10, 13] to be applicable to our more general problem, and we also develop new rules and bounds.

4 Computational results

Preliminary computational results on instances of small size show that the ASF formulation performs better than the TIF formulation only when the processing times of jobs are large (p_i chosen from $[1, 100]$) and n is very small. The TIF formulation is faster than the ASF formulation, however, when processing times are small (p_i chosen from $[1, 10]$).

We also observe that our B&B algorithm clearly outperforms both linear formulations. The SSDP algorithm solves instances in very short times when there are no precedence constraints. SSDP performs worse, however, when the precedence graph is dense. The B&B algorithm performs better, on the other hand, as the density of the precedence graph is increased.

References

1. Akturk, M., Ozdemir, D.: An exact approach to minimizing total weighted tardiness with release dates. IIE Transactions **32**, 1091–1101 (2000)
2. Held, M., Karp, R.: A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics **10**(1), 196–210 (1962)
3. Hoogeveen, J., Velde, S.: Stronger lagrangian bounds by use of slack variables: Applications to machine scheduling problems. Mathematical Programming **70**, 173–190 (1995)
4. Jouglet, A., Baptiste, P., Carlier, J.: Handbook of scheduling: Algorithms, Models and Performance Analysis, chap. 13, Branch and Bound Algorithms for Total Weighted Tardiness. CRC Press, Boca Raton, FL, USA (2004)
5. Keha, A., Khowala, K., Fowler, J.: Mixed integer programming formulations for single machine scheduling problems. Computers & Industrial Engineering. **56**, 357–367 (2009)
6. Lenstra, J., Kan, A.R., Brucker, P.: Complexity of machine scheduling problems. In: Studies in Integer Programming, *Annals of Discrete Mathematics*, vol. 1, pp. 343–362. Elsevier (1977)
7. Nessah, R., Kacem, I.: Branch-and-bound method for minimizing the weighted completion time scheduling problem on a single machine with release dates. Computers & Operations Research **39**(3), 471–478 (2012)
8. Pan, Y., Shi, L.: Dual constrained single machine sequencing to minimize total weighted completion time. IEEE Transactions on Automation Science and Engineering **2**(4), 344–357 (2005)
9. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Springer (2008)
10. Potts, C., Van Wassenhove, L.: A branch and bound algorithm for the total weighted tardiness problem. Operations Research **33**(2), 363–377 (1985)
11. Talla Nobibon, F., Leus, R.: Exact algorithms for a generalization of the order acceptance and scheduling problem in a single-machine environment. Computers & Operations Research **38**(1), 367–378 (2011)
12. Tanaka, S., Fujikuma, S.: A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. Journal of Scheduling **15**, 347–361 (2012)
13. Tanaka, S., Sato, S.: An exact algorithm for the precedence-constrained single-machine scheduling problem. European Journal of Operational Research **229**(2), 345–352 (2013)

MISTA 2013

An efficient metaheuristic for the inventory orienteering problem and the single-vehicle cyclic inventory routing problem

Pieter Vansteenwegen • Manuel Mateo

1 Introduction

Inventory Routing Problems (IRP) are currently getting a lot of attention (e.g. [1-3]). Besides the routing parameters, inventory and handling costs at the customers are also taken into account. Until now, a vehicle routing problem for which no inventory variant has been considered is the Orienteering Problem (OP). The goal of the regular OP [4] is to maximize the total score, collected by visiting a selection of customers, without violating a time constraint. Every customer can be visited at most once and the travel times between the customers are given. For the Inventory Orienteering Problem (IOP), inventory and handling costs are added.

The IOP considers n potential customers ($i=1,...,n$), each with an inventory cost (I_i in euro/ton*hour), a handling cost (H_i in euro/delivery), a constant demand rate (D_i in tons/hour) and a fixed reward (R_i in euro/hour). The time required to travel between two customers i and j is given by t_{ij} and is considered constant. A maximum time limits the availability to visit the customers. Other aspects related to distribution taken into account are the fixed vehicle operating cost (ψ in euro/hour), the average speed of the vehicle (v in km/hour), the travel cost (δ in euro per km) and the vehicle capacity (Q in tons). It is assumed that every customer has an infinite inventory capacity. Traditionally the objective is to minimise the total cost, although the reward may change the optimization into maximising profit. This problem is almost the same as the Single-Vehicle Cyclic Inventory Routing Problem (SV-CIRP) [5,6,7]. The only difference is that the available time to visit customers is not limited explicitly in the SV-CIRP.

We think the IOP would be closer to model reality than the SV-CIRP, since the IOP can limit this cycle time to a given upper bound, e.g. a number of hours or a working day. Nevertheless, this upper bound will only make the IOP easier to solve than the SV-CIRP. Moreover, benchmark test instances are only available for the SV-CIRP. Therefore, we will focus on the SV-CIRP, knowing that the proposed solution technique can also solve the IOP.

Pieter Vansteenwegen
KU Leuven - Centre for Industrial Management, Traffic & Infrastructure
E-mail: Pieter.vansteenwegen@cib.kuleuven.be

Manuel Mateo
Universitat Politècnica Catalunya, Departament Organització d'Empreses
E-mail: Manel.mateo@upc.edu

An important property, increasing the complexity of the problem, is that the single vehicle is allowed to make multiple trips from the depot during one cycle.

This means we are now looking for a cyclic distribution plan for a single vehicle, starting and ending at a single depot, for a set *S* of selected customers. A key decision variable will be the cycle time *T*, .i.e. the time between two deliveries to each customer. The total travel time will be T^r. Based on this cycle time, the total cost (*Cost* in euro/hour) can be defined as:

$$Cost(T) = \psi + \frac{\sum_{i \in S} H_i}{T} + T * \frac{\sum_{i \in S} D_i * I_i}{2} + \frac{\delta * \nu * T^r}{T} - \sum_{i \in S} R_i \qquad (1)$$

Please note that this total cost includes four terms, respectively, the vehicle cost, the handling cost, the inventory cost, the travelling cost and the rewards of visiting customers.

The cycle time is lower and upper bounded. It must be not shorter than the shortest travel time required to visit all selected customers divided in multiple trips (*Tmin*) and is limited to *Tmax*, based on the capacity *Q* of the vehicle. For a selection of customers divided in multiple trips, the ideal cycle time would be the "economic-order-quantity" cycle time *Teoq* [3]:

$$Teoq = \sqrt{\frac{\delta * \nu * Tmin + \sum_{i \in S} H_i}{\sum_{i \in S} \frac{D_i * I_i}{2}}} \qquad (2)$$

However, *Teoq* can only be selected if it lies between *Tmin* and *Tmax*. If *Teoq* is shorter than *Tmin*, the optimal cycle time *Topt* will be *Tmin* and if *Teoq* is longer than *Tmax*, it will be *Tmax*. Otherwise, *Topt=Teoq*.

The IRP was first introduced by Bell et al. [8]. A number of comprehensive survey papers about the IRP and its variants are available [1, 9-11]. The SV-CIRP and some exact and (meta)heuristic solution approaches are discussed in [3, 6, 7, 12, 13]. In Zhong and Aghezzaf [12], an in-depth analysis of the SV-CIRP reveals that the objective function is non-smooth and non-convex with many local minima. Zhong [7] discusses in detail the complexity of the SV-CIRP and presents an efficient mathematical formulation. Benchmark instances are available in [3,6,7].

2 Iterated Local Search

The Iterated Local Search (ILS) framework was successfully implemented before to deal with the SV-CIRP [3] and a variant of the OP [14]. The general structure of ILS is described by Lourenço et al. [15]. Algorithm 1 gives an overview of our ILS implementation, which is completely different from the implementation of Zhong and Aghezzaf [3]. The most important functions of this algorithm are *Initialisation*, *Local Search*, *Perturbation*, *Acceptance* and *Stopping Criterion*. Nevertheless, it is interesting to first discuss two building blocks used by these functions: *Divide* and *Insert.*

Divide tries to determine, for a given selection of customers *S*, the best division in trips and the best sequence of visits in each trip. Remember that the single vehicle can make multiple trips from the depot during one cycle. This division in trips is based on solving a Capacitated Vehicle Routing Problem (CVRP), for an increasing number of trips. The CVRP is solved by using the parallel version of the Clarke and Wright Savings Algorithm [16]. The CVRP-solution is improved, if possible, to decrease the total *Cost*.

Insert adds an extra customer to a given solution, without changing the division of the customers in trips. One by one, all non-visited customers are considered for insertion and the customer leading to the highest decrease in *Cost* is inserted. Then again, all non-visited customers are considered until no more customers can be inserted. When testing a customer for insertion, all possible positions in all possible trips are considered. The best position in each

trip is selected based on the lowest increase in travel time, and the best trip to insert the customer is selected based on the highest decrease of *Cost*.

Algorithm 1: ILS for the SV-CIRP

NoImprove ← 0
Solution ← *Initialisation*
BestFound ← Solution
while *NoImprove* < *MaxNoImprove* **do**
    Solution ← Local Search (Solution)
    if *Solution better than BestFound* **then**
        BestFound ← Solution
        NoImprove ← 0
    else
        NoImprove=NoImprove+1
    end if
    Solution ← Acceptance Criterion(BestFound, Solution)
    Solution ← Perturbation(Solution)
end while
Return (BestFound)

Initialisation starts by selecting randomly half of the customers. Then, *Divide* is applied in order to determine the best division in trips. If this division has a negative *Cost*, meaning it is worthwhile for the vehicle to be used for this tour, an initial solution is found. If this is not the case, again half of the customers are removed and *Divide* is applied again, until a solution with a negative *Cost* is found.

In *Local Search*, a heuristic considers one by one, all non-visited customers for insertion and the best one, decreasing *Cost* most, is inserted. Then again, all non-visited customers are considered until no more customers can be inserted. This heuristic alternates between two ways to look for the best non-visited customer: *Divide* (after adding the "non-visited customer" to the current set) and *Insert*. The most important advantage of this alternation is a high saving of computation time compared to using only *Divide*.

Perturbation evaluates a different combination of customers each iteration. It will remove *Nr* of subsequent customers from the current solution, starting at "position" *Pos*. The parameter *Nr* is initialised to one and is increased by one every iteration. *Nr* is limited to a given percentage of the customers in the current solution. If it becomes larger, it is reset to one. Similarly, *Pos* is initialised to one and is increased by *Nr* every iteration. If *Pos* would be larger than the number of customers in the solution, it is reduced by the number of customers. As a result of this setting of *Pos* and *Nr*, a different combination of customers is evaluated every iteration. This is one of the crucial elements in order to obtain high quality solutions. After the perturbation, the local search heuristic is applied to find a new local optimum.

The *Acceptance Criterion* typically has two extreme implementations [15]: with "random walk", the search always continues from the current solution, regardless its *Cost*; with "better" the search always continues from the best solution (*BestFound*). Our algorithm uses something in between. As a result, the algorithm is able to spend enough effort to further improve the best found solution. But when the number of iterations without improvement increases, the algorithm can spend more and more time on diversification in order to try to improve the current solution.

The algorithm stops when the number of subsequent iterations without improvement (*NoImprove*) reaches the value of *MaxNoImprove*. After testing several values, the following results were obtained with *MaxNoImprove* =200.

3 Experimental Results

The experimental results are based on five sets of ten benchmark instances for the SV-CIRP [3, 6, 7]. We made all sets available here:
http://www.mech.kuleuven.be/en/cib/op/opmainpage#section-21.

The results are summarised in Table 1. Our algorithm is compared with the best available metaheuristic for the SV-CIRP, called "Zho", of Zhong and Aghezzaf [3]. Our ILS algorithm was coded in Java and all experiments were executed on a Dell Latitude E5410 notebook, with Intel Core i5 2.40GHz processor and 4.0GB of RAM. The specifications for "Zho" can be found in [3] and [7].

The first two columns of Table 1 present the name of the set and the number of customers in these instances. The third one indicates the number of instances where our algorithm found better solutions than "Zho". The fourth column gives the average gap (in %) between the two approaches, calculated in this way: Gap = $100*(Cost_{Zho} - Cost_{ILS})/Cost_{Zho}$. Thus, a negative gap means that our approach outperforms "Zho". Finally, the fifth and sixth columns show the average CPU time of both solution methods.

Set	n	Better instances	Average gap (%)	CPU of ILS (s)	CPU of Zho (s)
1	15	3	-0.32	2	3
2	20	10	-21.62	4	212
3	25	2	0.67	28	106
4	30-67	10	-32.71	86	896
5	30-67	10	-26.22	221	1666
All	**15 - 67**		**-16.04**	**68**	**577**

Table 1: Summary of experimental results

Our ILS algorithm clearly outperforms the best technique currently available on all sets of instances, except for the instances of Set 3. For five of these instances, Zho performs better than our ILS. For three instances, both procedures reach the same results and for two instances our ILS performs better. When analysing these ten instances, it appeared to us that for these instances, it is very easy (and worthwhile) to visit all customers. The only decision that needs to be made is how these customers should be divided in trips in order to optimise the cycle time and the total cost; no selection is necessary. Zho focuses much more on that aspect and succeeds in obtaining a better division in trips than our ILS. Nevertheless, for two instances of Set 3, our ILS obtains new best known solutions.

When considering all 50 instances, our ILS obtains on average a *Cost* that is 16.04% better, and it is significantly faster. 32 new best known solutions are found. The quality improvement and the computation time reduction are the most noticeable for the larger instances of Set 4 and 5. Nevertheless, for all ten instances of only 15 customers (Set 1), the solutions obtained by our algorithm are the optimal solutions, in only 2 seconds on average. Only for Set 1 optimal solutions are available [6].

According to us, the main reason we outperform Zho is that we have a stronger diversification during the search process. We remove a changing number of customers in each diversification phase while Zhong and Aghezzaf [3] only relocate customers by swapping pairs of customers from different tours. Furthermore, we exploited some typical characteristics of the SV-CIRP.

4 Conclusions

In this paper, a fast and effective metaheuristic to deal with the inventory orienteering problem (IOP) as well as the single-vehicle cyclic inventory routing problem (SV-CIRP) is presented. One of the characteristics to consider it a very challenging optimisation problem is the fact that the single vehicle can make multiple trips. As a result, not only a selection of customers needs to be made, but the selected customers should also be divided in trips in an optimal way. We

developed an efficient Iterated Local Search (ILS) metaheuristic by exploiting this problem's characteristics.

The experimental results clearly show that our algorithm outperforms the existing solution approaches. On average, our ILS improves the results of Zhong and Aghezzaf [3] by 16.04% and 32 new best known solutions are found and reported. For the twenty largest instances with 30 to 67 customers, our approach improves the results with 29.46% and is eight times faster.

Moreover, the fact that our implementation of the ILS framework obtains much better results than the implementation by Zhong and Aghezzaf [3] clearly shows that different implementations of the same framework can lead to significantly different solution algorithms and that the performance obtained by any given framework depends strongly on the specific implementation of that framework.

References

1. Andersson H, Hoff A, Christiansen M, Hasle G, Lokketangen A, Industrial Aspects and Literature Survey: Combined Inventory Management and Routing, Computers & Operations Research, 37, 1515–36 (2010)
2. Schmid V, Doerner K, Laporte G, Rich routing problems arising in supply chain management, European Journal of Operational Research, 224, 435-48 (2013)
3. Zhong Y, Aghezzaf E-H, Effective Local Search Approaches for the Single-Vehicle Cyclic Inventory Routing Problem, International Journal of Services Operations and Informatics, accepted for publication (2013)
4. Vansteenwegen P, Souffriau W, Van Oudheusden D, The orienteering problem: a survey, European Journal of Operational Research, 209,1-10 (2011)
5. Christopher M, Logistics and Supply Chain Management: Strategies for Reducing Cost and Improving Service. Financial Times-Prentice Hall, UK (1998)
6. Aghezzaf E-H, Zhong Y, Raa B, Mateo M, Analysis of the single-vehicle cyclic inventory routing problem, International Journal of Systems Science, 43, 2040-9 (2012)
7. Zhong Y, Exact and Heuristic Methods for the Cyclic Inventory Routing Problem with Side-Constraints. PhD dissertation 2012, Ghent University, Ghent, Belgium
8. Bell WJ, Dalberto LM, Fisher ML, Greenfield AJ, Jaikumar R, Kedia P, et al., Improving the distribution of industrial gases with an on-line computerized routing and scheduling optimizer, Interfaces, 13, 4–23 (1983)
9. Coelho LC, Cordeau J-F, Laporte G, The inventory-routing problem with transhipment, Computers & Operations Research, 39, 2537–48 (2012)
10. Kleywegt AJ, Nori VS, Savelsbergh MWP, The Stochastic Inventory Routing Problem with Direct Deliveries, Transportation Science, 36, 94–118 (2002)
11. Adelman D, A Price-Directed Approach to Stochastic Inventory/Routing, Operations Research, 52, 499–514 (2004)
12. Zhong Y, Aghezzaf E-H, Combining DC-programming and steepest-descent to solve the single-vehicle inventory routing problem, Computers and Industrial Engineering, 61, 313-21 (2011)
13. Aghezzaf E-H, Raa B, Van Landeghem H, Modeling inventory routing problems in supply chains of high consumption products, European Journal of Operational Research, 169, 1048–63 (2006)
14. Vansteenwegen P, Souffriau W, Vanden Berghe G, Van Oudheusden D, Iterated Local Search for the Team Orienteering Problem with Time Windows, Computers & Operations Research, 36, 3281-90 (2009)
15. Lourenço H, Martin O., Stützle T, Iterated Local Search. In: Glover F, Kochenberger G, editors. Handbook of Metaheuristics, 321-53. Kluwer Academic Publishers, Dordrecht (2003)
16. Clarke G, Wright JW, Scheduling of vehicles from a central depot to a number of delivery points, Operations Research, 12, 568-81 (1964)

MISTA 2013

A heuristic approach to an integrated personnel rostering and task assignment problem

Pieter Smet · Greet Vanden Berghe

1 Introduction and related literature

Even after several years of academic research, automated personnel rostering remains both a relevant and an active research domain. The majority of the academic literature on personnel rostering deals with shift scheduling in which shifts are assigned to employees [1]. Task assignment is seldom integrated in the shift scheduling problem due to the increase in complexity it induces. However, as was shown by Maenhout and Vanhoucke [8], solving such problems in an integrated way can greatly improve the efficiency by which resources are deployed. The benefit of integrating various decisions into a single problem is also shown to be successful for many other problem domains, e.g. scheduling and routing [5] or packing and scheduling [10].

The present paper addresses the aforementioned issue by considering an integrated problem which combines personnel rostering and task assignment. The goal is to construct a schedule capable of showing *when* employees need to work and *which tasks* they should perform during their working shifts. In practice, these decisions are often taken in two separate stages. First, the employees' shift schedules are constructed. Later, usually at the start of each shift, tasks are assigned by a team leader in an ad hoc manner. This approach often causes tasks to be left unassigned due to a lack of personnel in a particular shift. This either affects the future planning, possibly delaying subsequent operations, or leads to additional operational expenses in the form of temporary workers being called in last minute. These issues give rise to the need for a new approach to the integrated problem of personnel rostering and task scheduling.

Only few papers discuss problems which integrate task assignment and personnel rostering. Detienne et al. [3] solve a related timetabling problem which involves two decision stages. First, they determine when each employee should work. In the second

Pieter Smet
CODeS, KAHO Sint-Lieven
Gebroeders De Smetstraat 1, 9000 Gent, Belgium
E-mail: pieter.smet@kahosl.be

Greet Vanden Berghe
CODeS, KAHO Sint-Lieven
Department of Computer Science, KU Leuven
E-mail: greet.vandenberghe@kahosl.be

phase, for each working period, the used qualification for each employee is decided on. Guyon et al. [6] build upon this work by including the decision of the start time of activities. The integrated employee timetabling and production scheduling problem is solved using different decomposition approaches. The problem perhaps most closely resembling the subject of the present study is discussed by Dowling et al. [4] in the context of a large international airport. First, the personnel rostering problem is solved heuristically for a long period. Afterwards individual tasks are allocated to the available employees on a day-to-day basis.

2 Problem description

Given are a set of tasks and a set of personnel with each employee having a contract. Each task is defined by fixed start and end times. Their assignment to personnel is further restricted by qualification requirements. A task can only be assigned to an employee if that employee is assigned to a working shift that covers the task's execution times. The contract of each employee contains constraints from three generic classes of soft constraints: *counters*, *series* and *successive series* [2]. These constraints model the different workforce related restrictions that stem from both legislation and personal preferences. The assignment of shifts to employees is restricted by these constraints. The workforce typically exhibits a large degree of heterogeneousness due to different contracts (e.g. full time, 75% part time, 45% part time, etc.) Furthermore, employees have a set of qualifications, each of which can be given a priority. The scheduling horizon for this problem is typically one or four weeks, depending on the context.

The goal is to 1) cover the task-demand and 2) construct rosters for employees that violate as few as possible of the contractual constraints.

This problem resembles the standard personnel rostering problems [9], however, here, the coverage requirements are not explicitly defined for a *shift-skill* combination. In the present study, the tasks to be performed implicitly delineate the shift coverage requirements based on their start and end times.

3 Two phase heuristic algorithm

Our solution approach involves two consecutive phases. First, tasks are placed in shifts, which are then heuristically assigned to the available workforce members according to their contracts.

When allocating the tasks in shifts, the objective is to minimise the required number of shifts, consequently minimising the number of required employees. By taking into account this objective, the issue of additional personnel costs, e.g. caused by deploying temporary workers, is minimised. This problem is very similar to the SMPTSP [7]. However, the present problem does not define an upper bound on the number of available shifts. Moreover, it is possible that there are overlapping shifts such that tasks can feasibly be assigned to multiple shifts.

In the second phase, the solution of the first phase (the shifts required to perform all tasks) is used to construct the employees' shift schedules. This second phase corresponds to solving a standard personnel rostering problem since the first phase converted the defined tasks to shift coverage requirements. Due to the complex contracts, we have opted for a heuristic approach to the resulting personnel rostering problem.

Indeed, because the contractual constraints are defined as soft constraints, they may get violated. In practice however, this does not pose a problem since it reflects real world situations.

4 Discussion

Organisations require an efficient deployment of their staff to reduce operational expenses caused by ineffective use of resources. In automated personnel rostering approaches, typically, only the assignment of shifts or days-off is considered, while a more fine-grained allocation could benefit both the staff members and the organisation. In the present study we discuss an integrated personnel rostering and task scheduling problem in which the goal is to construct a schedule that shows *when* employees need to work and *which tasks* employees should perform in their working shifts.

We present a two-phase heuristic algorithm capable of generating such schedules for real-world scenarios in which various workforce related constraints are of importance. Besides respecting the multitude of contractual constraints, the presented approach focusses on the efficient deployment of the available workforce by minimising the required number of personnel necessary to execute all tasks. This is achieved by allocating the tasks in shifts in the first phase. An efficient metaheuristic algorithm which then tries to assign these shifts in the best possible manner allows for the fast generation of a valid schedule for both the organisation and the employees.

Acknowledgements This research was carried out within the IWT 110257 project.

References

1. Van den Bergh, J., Beliën, J., De Bruecker Philippe andDemeulemeester, E., De Boeck, L.: Personnel scheduling: A literature review. European Journal of Operational Research **226**(3), 367 – 385 (2013)
2. Bilgin, B., De Causmaecker, P., Rossie, B., Vanden Berghe, G.: Local search neighbourhoods to deal with a novel nurse rostering model. Annals of Operations Research **194**(1), 33–57 (2012)
3. Detienne, B., Peridy, L., Pinson, E., Rivreau, D.: Cut generation for an employee timetabling problem. European Journal of Operational Research **193**(3), 1178–1184 (2009)
4. Dowling, D., Krishnamoorthy, M., Mackenzie, H., Sier, D.: Staff rostering at a large international airport. Annals of Operations Research **72**, 125–147 (1997)
5. Eveborn, P., Rönnqvist, M., Einarsdóttir, H., Eklund, M., Lidén, K., Almroth, M.: Operations research improves quality and efficiency in home care. Interfaces **39**(1), 18 – 34 (2009)
6. Guyon, O., Lemaire, P., Pinson, E., Rivreau, D.: Cut generation for an integrated employee timetabling and production scheduling problem. European Journal of Operational Research **201**(2), 557–567 (2010)
7. Krishnamoorthy, M., Ernst, A., Baatar, D.: Algorithms for large scale shift minimisation personnel task scheduling problems. European Journal of Operational Research **219**, 34–48 (2012)
8. Maenhout, B., Vanhoucke, M.: An integrated nurse staffing and scheduling analysis for longer-term nursing staff allocation problems. Omega **41**(2), 485–499 (2013)
9. Smet, P., Bilgin, B., De Causmaecker, P., Vanden Berghe, G.: Modelling and evaluation issues in nurse rostering. Annals of Operations Research (in press). DOI 10.1007/s10479-012-1116-3
10. Verstichel, J., Berghe, G.: A late acceptance algorithm for the lock scheduling problem. In: S. Voss, J. Pahl, S. Schwarze (eds.) Logistik Management, pp. 457–478. Physica-Verlag HD (2009)

MISTA 2013

Dual Discounting Functions for Internet Shopping Optimization Problem

Jacek Blazewicz · Pascal Bouvry · Jedrzej Musial · Johnatan E. Pecero · Denis Trystram

1 Introduction

Electronic Commerce is definitely one of the youngest computer science branches. Moreover, we could claim that apart from being a cutting-edge discipline, it is also one of the fastest developing parts of computer science. Revenue of the e-commerce market grows rapidly. One can say that the multidisciplinary nature (connecting many of the well-known computer science branches like operations research, combinatorial optimization, algorithms, with other sciences - e.g. logistics, marketing, and many others) lies behind the success of the e-commerce science. One of the very important topics of research in that area is, motivated by practical applications, Internet shopping which is becomming more and more popular with every upcoming year. Products available in online stores are often cheaper than the ones offered by regular local retailers and a wide choice of offers is available just a click away from the customer. A crucial aspect of online shopping is the time spent on comparing offers and a comfortable way of shopping regardless of shop location.

Jacek Blazewicz, Jedrzej Musial
Institute of Computing Science
Poznan University of Technology
ul. Piotrowo 2
60-965 Poznan
Poland
E-mail: Jedrzej.Musial@cs.put.poznan.pl

Pascal Bouvry, Johnatan E. Pecero
Computer Science and Communications Research Unit
University of Luxembourg
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg
Luxembourg

Denis Trystram
Grenoble Institute of Technology - ENSIMAG
51 avenue Kuntzmann
38300 Montbonnot St Martin
France

2 Internet Shopping Optimization Problem

Internet Shopping Optimization Problem (ISOP) tackles the situation when a single buyer is looking for a multiset of products $N = \{1, \ldots, n\}$ to buy in m shops, $M = \{1, \ldots, m\}$. A multiset of available products N_i, a cost p_{ij} of each product $j \in N_i$, and a delivery cost d_i of any subset of the products from the shop to the buyer are associated with each shop $i \in M$, $i = 1, \ldots, m$. It is assumed that $p_{ij} = \infty$ if $j \notin N_i$. The problem lies in finding a sequence of disjoint selections (or carts) of products $X = (X_1, \ldots, X_m)$, which we call a *cart sequence*, such that $X_i \subseteq N_i$, $i = 1, \ldots, m$, $\cup_{i=1}^{m} X_i = N$, and the total product and delivery cost, denoted as $F(X) := \sum_{i=1}^{m} \left(\delta(|X_i|) d_i + \sum_{j \in X_i} p_{ij} \right)$, is minimized. Here $|X_i|$ denotes the cardinality of the multiset X_i, and $\delta(x) = 0$ if $x = 0$ and $\delta(x) = 1$ if $x > 0$. We denote this problem as ISOP (Internet Shopping Optimization Problem), its optimal solution as X^*, and its optimal solution value as F^*. Follow [2,4,3] for detailed formulations, NP-hardness proof and heuristic algorithm definitions. Moreover, you can find a detailed description of a computational experiment and comments on its results. The most interesting (and the most complicated) specialization of the ISOP is the so called Internet Shopping with Price Sensitive Discounts[1,3] problem. For each Internet shop, standard prices for the products are known, as well as an increasing discounting function of total standard and delivery price. Buying all the reacquired products at the minimum total discounted price constitutes the problem. Its mathematical program can be written as follows:

$$\begin{aligned} \min \ & \sum_{i=1}^{m} f_i\Big(\sum_{j \in N_i} p_{ij} x_{ij}\Big) + \sum_{i=1}^{m} d_i y_i, && (1) \\ s.t. \ & \sum_{i \in M} x_{ij} = 1, \ j = 1, \ldots, n, \\ & 0 \le x_{ij} \le y_i, \ i = 1, \ldots, m, \ j = 1, \ldots, n, \\ & x_{ij} \in \{0, 1\}, \ y_i \in \{0, 1\}, \ i = 1, \ldots, m, \ j = 1, \ldots, n. \end{aligned}$$

where $f_i(T)$ - discounting function for the final sum of standard price T in shop i at all points $T > 0$, $f_i(0) = 0$.

3 Two price sensitive functions for ISOP

During MISTA conference a new optimization problem will be introduced - Dual Discounting Functions for Internet Shopping Optimization Problem. Moreover, its mathematical formulation as well as the first computational experiment results will be presented. Price sensitive shipping costs are often used in Internet shops to attract customers and to encourage them to buy more products. A typical example is the following advertisement in an Internet shop selling books and CD's: If the value of your purchase is at least 25 euros, then we will ship products for a half of shipping cost; 35 euros - a half of price shipped by courier; 40 - free post office shipping; 50 - free courier shipping. Values and thresholds can vary depending on the seller. One can describe ISOP with two price sensitive functions as follows. We would like to buy a number of products (n). Products are offered by many stores (m) from different locations - some are distant, some are close to our position. The price of each product n in every store m could

Table 1 Example price structure for a delivery.

Delivery type	Payment method	$T_i < 25$	$25 \leq T_i < 50$	$50 \leq T_i < 75$	$T_i \geq 75$
Post office	Credit card	10	5	0	0
Post office	At pick-up	15	10	5	0
Courier	Credit card	15	15	5	0
Courier	At pick-up	20	20	10	0

be written as p_{ij} where $i = 1, \ldots, m,\ j = 1, \ldots, n$. To every store m corresponds a discounting function (on the base of the total cost of the shopping from the one store) $f_i(T_i)$. Our goal is to find out how to pay the lowest bill for all the products including delivery costs. Mathematical program formulation can be written as follows:

$$\begin{aligned} \min\ & \sum_{i=1}^{m} f_i(\sum_{j \in N_i} p_{ij} x_{ij}) + \sum_{i=1}^{m} d_i(\sum_{j \in N_i} p_{ij} x_{ij}), \\ s.t.\ & \sum_{i \in M} x_{ij} = 1,\ j = 1, \ldots, n, \\ & x_{ij} \in \{0, 1\},\ i = 1, \ldots, m,\ j = 1, \ldots, n. \end{aligned} \tag{2}$$

The discounting function could look as follows (an example):

$$f_i(T_i) = \begin{cases} T_i & \text{if } 0 < T_i \leq 50, \\ 50 + 0.95(T_i - 50) & \text{if } 50 < T_i \leq 100, \\ 50 + 0.95 * 50 + 0.9(T_i - 100) & \text{if } 100 < T_i \leq 150, \\ 50 + 0.95 * 50 + 0.9 * 100 + 0.85(T_i - 150) & \text{if } T_i > 150. \end{cases}$$

where T_i is the total standard price of books selected in bookstore i. The shipping cost function is more complicated for its shape depends additionally on a customer's further choices (delivery type - e.g. by post office, courier; payment method). We can define it as $d_i(T_i, dv_1, dv_2), i = 1, \ldots, m$. Therefore, for the present discussion we can state that a customer makes two decisions - delivery type (dv_1) and payment method (dv_2). Shipping costs function example is shown in Table 1.

To solve this problem one can use some of the already prepared heuristics, however, before finishing computational experiments it cannot be said that they will be as efficient as for IS with Price Sensitive Discounts problem. Moreover, we can presume that new specific algorithms for this interesting case should be developed.

Acknowledgements The research was partially supported by the NCN grant N N519 643340.

References

1. Blazewicz J., Bouvry P., Kovalyov M.Y., Musial J., Internet Shopping with Price Sensitive Discounts, *4OR- A Quarterly Journal of Operations Research*, DOI: 10.1007/s10288-013-0230-7, (2013).
2. Blazewicz J., Kovalyov M.Y., Musial J., Urbanski A.P., Wojciechowski A., Internet Shopping Optimization Problem, *International Journal of Applied Mathematics and Computer Science*, Vol. 20(2), 385-390 (2010).
3. Musial J., Applications of Combinatorial Optimization for Online Shopping, NAKOM, Poznan (2012).
4. Wojciechowski A., Musial J., Towards Optimal Multi-item Shopping Basket Management: Heuristic Approach, *Lecture Notes in Computing Science*, Vol. 6428, 349-357 (2010).

MISTA 2013

Complexity analysis of the discrete sequential search problem with group activities

Kris Coolen · Roel Leus · Fabrice Talla Nobibon

1 Introduction

This paper studies the computational complexity of the *discrete sequential search problem with group activities* defined by Wagner and Davis [13], in which a single object is hidden in one of n boxes and the probability that box k contains that object is π_k. The boxes are searched one at a time and the cost of searching box k is t_k. If the box containing the object is searched then the object is detected with certainty. There are also m 'group activities'. Each group activity ℓ has a cost R_ℓ and is associated with a subset $S_\ell \subset \{1, \ldots, n\}$ of boxes. Notice that some boxes may appear in more than one subset S_ℓ. The group activities are said to be *conjunctive* if any box can be searched only when all the group activities in which it appears have been performed whereas for *disjunctive* group activities, a box can be searched as soon as at least one of the group activities in which it appears has been executed. The goal is to find a sequence (defining a search strategy) in which the boxes are to be searched and the group activities are to be performed so as to minimize the expected cost while satisfying the precedence constraints imposed by the group activities. We refer to the discrete sequential search problem with exclusively conjunctive, respectively disjunctive, group activities as Problem 1, respectively Problem 2.

Discrete sequential search problems have applications in various areas such as quality control [9], research and development [6], diagnostic tests on components of complex radar, missile, and communications systems [3], and can be placed in a more general framework of bandit processes to which Gittins' Index Theorem [2] applies. In the diagnostic sequencing problem, an example of a group activity would be a task such as removing an access cover, draining fluids, disconnecting a power supply, etc., that must occur before a set of components can be tested. In the problem of sequencing tasks in a research-and-development project, a group activity may represent a facility that must be constructed before a set of tasks can be completed [13].

In 2001, Wagner and Davis [13] presented an integer-programming model for the discrete sequential search problem including both conjunctive and disjunctive group activities. Based on their experiments, they conjectured that the conjunctive case (Problem 1) can be solved as a linear program. Recently, in [12] we have described

ORSTAT, Faculty of Economics and Business, KU Leuven, Belgium
E-mail: < $first_name.last_name$ >@kuleuven.be

a counterexample for which the optimal value of the linear program proposed by Wagner and Davis is different from the optimal value of the integer-programming model, hence contradicting the conjecture.

In this paper, we observe that Problem 1, respectively Problem 2, is equivalent to scheduling a set of jobs on a single machine to minimize the total weighted completion time with *and*, respectively *or*, precedence constraints, described by a special bipartite graph. We also establish NP-hardness results for Problem 1 and Problem 2, and distinguish some polynomially solvable cases.

2 Link with scheduling literature

The discrete sequential search problem without group activities is equivalent to a single-machine scheduling problem, where each box i corresponds with a job i that is characterized by a weight $w_i \equiv \pi_i$ and a processing time $p_i \equiv t_i$ [7]. The machine can process at most one job at the time and preemption is not allowed. The goal of this problem is to find a sequence of jobs that minimizes the total weighted completion time $\sum_{i=1}^{n} w_i C_i$, where C_i is the completion time of job i. In standard notation [4] the problem is referred to as $1||\sum w_i C_i$. This generic problem has an $O(n \log n)$ algorithm based on Smith's rule [11] that schedules jobs in such a way that for all pairs of jobs i and j, job i is executed before job j if $p_i w_j < p_j w_i$. The problem $1||\sum w_i C_i$ has been extended following several directions including the addition of precedence constraints among jobs. When precedence constraints are included, the problem is written as $1|prec|\sum w_i C_i$. In the literature, precedence constraints are specified by means of a directed acyclic graph $G = (V, A)$, where the set V of vertices corresponds with the set of jobs and an arc $(i, j) \in A$ indicates that job i must be executed before job j.

A bipartite graph $G = (V, A)$ is such that V can be partitioned into two subsets V_1 and V_2 with each arc in A going from a vertex in V_1 to a vertex in V_2. Lenstra and Rinnooy Kan [8] show that $1|prec|\sum w_i C_i$ is strongly NP-hard when G is an arbitrary bipartite graph, even if each job has a unit processing time. For ease of exposition, when the precedence constraints are represented by a bipartite graph $G = (V_1 \cup V_2, A)$ we write $1|V_1 \cup V_2|\sum w_i C_i$. Polynomially solvable cases have been established by Horn [5] and Sidney [10], who present an $O(n \log n)$ algorithm for the case where G is a rooted tree, and by Adolphson [1], who describes an $O(n \log n)$ algorithm for the case where G is a series-parallel graph.

Problem 1 is a special case of $1|V_1 \cup V_2|\sum w_i C_i$. Indeed, consider an instance of the former problem with n boxes and m group activities. We construct an equivalent instance of $1|V_1 \cup V_2|\sum w_i C_i$ with $m + n$ jobs, where the first m jobs, called *group-activity jobs*, correspond with the m group activities and belong to V_1 whereas the last n jobs, called *box jobs*, correspond with the n boxes and are all in V_2. The weight w_i of group-activity job i $(i = 1, \ldots, m)$ is 0 whereas the weight w_i of box job i $(i = m + 1, \ldots, m + n)$ is the probability π_{i-m} that the object is in the box $i - m$. Next, the processing time p_i of a group-activity job i $(i = 1, \ldots, m)$ is exactly the cost R_i of group activity i whereas the processing time p_i of box job i $(i = m + 1, \ldots, m + n)$ is equal to the cost t_{i-m} of searching box $i - m$. Finally, the bipartite precedence graph $G = (V_1 \cup V_2, A)$ is such that there is an arc from group-activity job i $(i = 1, \ldots, m)$ to box job j $(j = m + 1, \ldots, m + n)$ if and only if box $j - m$ belongs to S_i (the subset of boxes associated with group activity i). This construction is done in polynomial time with respect to the size of the instance (in this case $n + m$). Notice that each

job in V_1 has a weight of 0 whereas the weight of a job in V_2 can be any arbitrary (non-negative) number. Therefore, we denote the corresponding scheduling problem by $1|V_1 \cup V_2, (0, free)| \sum w_i C_i$, which is a special case of $1|V_1 \cup V_2| \sum w_i C_i$. For disjunctive group activities (Problem 2), the same construction may be applied. However, a job in V_2 can be started as soon as at least one of its predecessors in V_1 has been executed (*or*-type precedence constraint).

3 Results

Theorem 1 *Both Problem 1 and Problem 2 are strongly NP-hard, even when all group activities have equal costs and all boxes are identical.*

Note that, with identical boxes, we mean that all boxes have equal costs as well as equal probabilities of containing the hidden object. NP-hardness for the above special case of Problem 1 can be shown by establishing the equivalence with $1|prec| \sum w_i C_i$. The result for Problem 2 follows from a reduction from a variant of 3-Dimensional Matching. From this we conclude that, unless P=NP, it is impossible that there exists any (compact-size) linear program for solving Problem 1, hereby generalizing the result in [12].

Furthermore, we show that the following special cases are polynomially solvable:

1. Problem 1 and Problem 2 when all boxes appear in at most one subset S_ℓ
2. Problem 1 and Problem 2 when $|S_\ell| = 1$ for all group activities ℓ
3. Problem 2 when all group activities have equal costs, all boxes are identical, and $|S_\ell| = 2$ for all group activities ℓ

References

1. D. Adolphson. Single machine job sequencing with precedence constraints. *SIAM Journal on Computing*, 6:40–54, 1977.
2. J.C. Gittins. *Multi-Armed Bandit Allocation Indices*. Wiley, 1989.
3. B. Gluss. An optimum policy for detecting a fault in a complex system. *Operations Research*, 7:468–477, 1959.
4. 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.
5. W.A. Horn. Single machine job sequencing with treelike precedence ordering and linear delay penalties. *SIAM Journal of Applied Mathematics*, 23(2):189–202, 1972.
6. W.B. Joyce. Organization of unsuccessful R&D programs. *IEEE Transactions on Engineering Management*, 18(2):57–65, 1971.
7. F.P. Kelly. A remark on search and sequencing problems. *Mathematics of Operations Research*, 7(1):154–157, 1982.
8. J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35, 1978.
9. L.G. Mitten. An analytical solution to the least cost testing sequence problem. *Journal of Industrial Engineering*, 1(11):17–17, 1960.
10. J.B. Sidney. Decomposition algorithms for single machine scheduling with precedence relations and deferral costs. *Operations Research*, 23(2):283–298, 1975.
11. W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.
12. F. Talla Nobibon, K. Coolen, and R. Leus. A note on 'discrete sequential search with group activities'. *Decision Sciences*, forthcoming.
13. B.J. Wagner and D.J. Davis. Discrete sequential search with group activities. *Decision Sciences*, 32(4):557–573, 2001.

MISTA 2013

An exact procedure for an integrated project staffing problem

Broos Maenhout · Mario Vanhoucke

1 Introduction

Labour costs comprise a vast amount of a project's expenditure as projects are typically very labour intensive. Personnel resources are often seen as aplenty available and little attention is spent on the calendar constraints and on the related personnel scheduling problem. Solely focusing on the project scheduling leads to suboptimal decisions and often results in ineffective and poor schedule outcomes especially for these personnel resources. Hence, when scheduling projects under resource constraints, assumptions are typically made with respect to this resource availability. In personnel rostering problems, on the other hand, important assumptions are made with respect to the staffing requirements. It is obvious that the scheduling of activities is a driver for the work demand pattern and/or that the project schedule is aligned according to the employed personnel scheduling policies and practices. In this research we aim to integrate these two scheduling problems.

2 Problem description

Project staffing and scheduling are two isolated topics that are widely studied in the literature in various guises and formulations. Relevant resource-constrained project scheduling models where the resources may possess the characteristics of personnel resources, are the well-known resource-constrained project scheduling problem, the resource availability cost problem [11], the time-constrained project scheduling problem (TCPSP) [6] and the resource renting problem (RRP) [12]. For an overview we refer to [9].

Broos Maenhout
Faculty of Economics and Business Administration, Ghent University, Belgium
E-mail: Broos.Maenhout@Ugent.be

Mario Vanhoucke
Faculty of Economics and Business Administration, Ghent University, Belgium
Vlerick Business School, Belgium
University College London, United Kingdom
E-mail: Mario.Vanhoucke@Ugent.be

The relevant literature on the integrated project staffing and personnel scheduling problem is limited. [1] and [2] use integer programming to minimize the project makespan and personnel costs. [5] discuss an exact optimization method for a multi-skill project scheduling problem. The problem is then to schedule the project activities given the personnel schedule such that the scheduled personnel is able to carry out the different project activities. [8] presents a MIP model for the simultaneous scheduling and staffing of multiple projects with multiple resources. Several other related papers involve the audit scheduling problem where tasks have to be assigned to a set of workers that have an overall capacity for the entire planning horizon (cfr. [7] for an overview).

We study a strategic budgeting problem that determines the project staffing plan and corresponding personnel schedule for a single project. The problem under study integrates the consecutive planning levels of project scheduling and project staffing. We decide on the best project schedule (i.e. the starting times of the activities) and corresponding staff composition in terms of resource type use (regular workers, overtime and temporal workers) to carry out a single project with the objective of cost minimization.
The activity and project scheduling characteristics can be stated as follows. We assume that we have a project network with a set of activities and a set of direct precedence relations. These activities are to be scheduled without pre-emption on a set of renewable personnel resource types. Each activity has a deterministic duration and requires a number of man units per time unit. We assume that there is a prescribed maximum project duration. The available set of resources for each resource type is determined in the personnel scheduling step for each time unit of the project horizon. A schedule is said to be feasible if it is non-preemptive and if the project duration, precedence and resource constraints are satisfied. The processing of the project activities requires renewable personnel resources. We consider several types of personnel resources: regular personnel time units, overtime units and temporal personnel time units. These personnel resources have to be scheduled such that all activities can be processed according to the project schedule. The underlying homogeneous personnel scheduling problem for the regular workers is a manpower days-off scheduling problem [1,3]. All personnel members are anonymous and possess all relevant skills to carry out the activities. In the strategic manpower scheduling problem we have to decide on the total number of workers, overtime and temporary help.

3 Procedure

The solution approach is an exact branch-and-price procedure. The problem formulation is decomposed into a master program and a subproblem, which formulation defines the structure of a (feasible) personnel line-of-work [4]. The decision variables of the master problem are the binary activity start time variables and integer personnel column variables. Branch-and-price solves the linear program of the master problem by relaxing the integrality constraints via a column generation procedure. Column generation iterates between the restricted master problem and a personnel scheduling pricing problem, which takes dual information from the master problem into account. Since adding a personnel or workload schedule can decrease the value of the linear programming solution, we solve the pricing problem that aims to find a schedule with the

largest negative reduced cost. If this minimum value is nonnegative, we have found the optimal solution of the linear programming relaxation. When this optimal LP relaxation of the master problem has not an integral solution, we apply a branch-and-bound search by making use of dedicated branching rules.

4 Computational experiments

We have tested our procedure on a randomly generated dataset under a controlled design consisting up to 30 activities. Based on this dataset, we give insight in the computational performance of the proposed procedure. We compare the performance of different branching schemes, analyze the impact of different node reduction mechanisms and compare the performance of the proposed algorithm with several other techniques. Further, we examine the impact of the activity and project characteristics, the personnel staffing characteristics and the objective function weights for this integrated project staffing problem in order to provide computational and managerial insights.

References

1. Alfares, H. and Bailey, J., Integrated project task and manpower scheduling, IIE Transactions, 29, 711-717 (1997)
2. Alfares, H., Bailey, J. and Lin, W., Integrated project operations and personnel scheduling with multiple labour classes, Production Planning and Control, 10, 570-578 (1999)
3. Alfares, H., Efficient optimization of cyclic labor days-off scheduling, OR Spektrum, 23, 283-294 (2001)
4. Barnhart, C., Johnson, E., and Nemhauser, G., Branch-and-price: Column generation for solving huge integer programs, Operations Research, 46, 316-329 (1998)
5. Bellenguez-Morineau, O., Methods to solve multi-skill project scheduling problem, 4OR, 6, 85-88 (2008)
6. Deckro, R. and Herbert, J., Resource constrained project crashing, Omega International Journal of Management Science, 17, 69-79 (1989)
7. Dodin, B. and Elimann, A., Audit scheduling with overlapping activities and sequence-dependent setup times, European Journal of Operational Research, 97, 22-33 (1997)
8. Heimerl, C. and Kolisch, R., Scheduling and staffing multiple projects with a multi-skilled workforce, OR Spectrum, 32, 343-368 (2010)
9. Herroelen, W., Demeulemeester, E., and De Reyck, B.,. Project scheduling - recent models, algorithms and applications, chapter A classification scheme for project scheduling problem, pages 126. Kluwer Academic Publishers 1, Boston (1999)
10. Hung, R. (1994). Single-shift off-day scheduling of a hierarchical workforce with variable demands. European Journal of Operational Research, 78:4957.
11. Mohring, R., Minimizing costs of resource requirements in project networks subject to a fixed completion time, Operations Research, 32, 89-120 (1984)
12. Nubel, H., The resource renting problem subject to temporal constraints, OR Spektrum, 23, 359-381 (2001)

MISTA 2013

A fast solution method for the Time-Dependent Orienteering Problem with time windows

C. Verbeeck ·
E.-H. Aghezzaf ·
P. Vansteenwegen

1 Introduction

This research focuses on developing an efficient solution method to solve the time-dependent orienteering problem with time windows (TD-OPTW), requiring only a few seconds of computation time.

In Orienteering problems an optimal combination of vertices (locations) needs to be selected and the routing between the vertices needs to be optimized. In the time-dependent version the travel time between two vertices depends on the departure time at the first vertex.

Orienteering problems [8] are typically used in logistic planning tools where each vertex resembles a customer and the score reflects the profit margin. Furthermore, they serve as the basic problem formulation of personalized touristic trip planners where a vertex stands for a point of interest and the score indicates the personal interest of the tourist. For a longer list of practical and real-life applications of the OP and its variants, we refer to the recent survey by Vansteenwegen et al [8]. The time-dependent problem formulation allows to tackle congestion related issues in routing problems such as morning and evening peaks on the highways or crowded city center traffic situations [5]. Secondly, multi-modal applications for logistic or touristic trip planners also rely on TD-OP solution methods [1, 3]. The most common example is the combination of walking and using public transport, where the waiting time at a bus station and the time table of the bus result in time-dependent travel times. Due to the rise in congestion problems on the one hand and the acceptance of mobile devices with GPS and internet connection on the other hand, the fast construction or update of routes

C.Verbeeck
Ghent University, Department of Industrial management
E-mail: Cedric.Verbeeck@ugent.be

E.-H. Aghezzaf
Ghent University, Department of Industrial management
E-mail: elhoussaine.aghezzaf@ugent.be

P. Vansteenwegen
KU Leuven, Centre for Industrial Management, Traffic and Infrastructure
E-mail: pieter.vansteenwegen@cib.kuleuven.be

based on congestion information increasingly becomes a necessity for a number of interesting business applications.

2 Methodology

Since high-quality solutions are required and the computation time should be limited to only a few seconds, the literature on vehicle routing suggests the implementation of a local search based metaheuristic [7]. These solution methods try to find better solutions combining efficient local search procedures with mechanisms to escape local optima. Three time-dependent local search procedures with a local evaluation metric have been successfully implemented. A local evaluation metric prevents a full and time-consuming evaluation of a solution after every modification attempt and enables an efficient checking and updating mechanism. Therefore, we store for every vertex in the current solution the maximum amount of time that a visit to it can be postponed before the solution becomes infeasible. Firstly, the problem-specific *insert* local search procedure iteratively attempts to insert non-included vertices into an existing solution. Secondly, a time-dependent *replace* procedure tries to replace a vertex from the current solution with a non included vertex with a higher score. Thirdly, the *swap* local search procedure tries to exchange two solution member vertices in order to save travel time.

The basic metaheuristic framework of an ant colony system (ACS) was used. This choice was motivated by the fact that generally very complex problems require simple solution frameworks. The ACS is based on the ant colony optimization algorithm (ACO) of Ke et al [4] and Schilde et al [6] for the OP and on the ACS of Donati et al [2] for the TD-VRP. The ACS starts by sequentially creating a number of initial solutions. The construction process starts from the first vertex and in succession adds a new vertex to the solution based on *greedy information* and *pheromone trails* of the arcs leading to that vertex. The greedy information is based on a ratio which consists of the score of the next vertex and the distance towards the next vertex. After the creation of a complete solution, the included arcs are made less attractive for the following solution construction procedures, in order to increase diversification. This is done by depreciating the pheromone values with an evaporation rate factor (*local pheromone update*). Once all solutions of one iteration are created, they are improved, using the following sequence of local search procedures in a first-improving manner until no more feasible improvements can be found: swap, insert and replace. The solution with the highest score is stored. Finally, the arcs that are used in this solution are made more attractive to be used in the solution construction procedure of succeeding iterations, by increasing their corresponding pheromone value. These steps are repeated during a specified number of iterations. To prevent that only a couple of arcs dominate this solution construction procedure, the pheromone values are reset during a *global pheromone update* when no improvement can be found during a certain number of iterations.

Realistic road information of the geographical region of Flanders in Belgium is used as provided by OpenStreetMap (OSM). Using this large amount of unstructured and partially redundant information would be too complicated to use in a metaheuristic so we filtered and efficiently restructured the information without sacrificing the quality of the road network. These steps resulted in a strongly connected graph representing a road network with 84 720 vertices and 116 683 arcs. Once the static road network is created, congestion information should be added. However, OSM offers no congestion information apart from the road types, so as preliminary solution we mapped each

OSM road type to a corresponding arc congestion category based on the authors insight. To lower the execution time of the travel time calculation, an adapted version of the shortest path algorithm of Dijkstra was implemented using binary heaps and the speed model of Donati et al [2] to calculate the time-dependent travel times on each arc. However, this implementation was still too costly to use during the execution of the ACS. Therefore, the travel time between the subset of vertices, representing the customers or depots on the graph, is calculated in a pre-processing phase. After these steps a new virtual network has been constructed. This network is implemented by a completed graph which consists of nodes representing the customers to visit and virtual arcs with time-independent and time-dependent travel time profiles which model the travel time behavior on a concatenation of arcs.

3 Results

To test the proposed algorithm, we created a set of 36 instances by randomly selecting respectively 20, 50, 100 and 250 vertices in our OSM road network and generating 3 variations in maximum allowed travel time and 3 variations in time window severity. Subsequently, all instances were solved first as time-*independent* OPs using the commercial solver CPLEX. During this optimization, the travel time is calculated using, on each arc, the maximum speed of the corresponding arc category. Following this optimization, the optimal time-*independent* solution is used to create a known optimal solution by slightly modifying the time-dependent instance. For each virtual arc included in the optimal time-independent solution, the travel speed is set to its maximal value, but only during the period in time this virtual arc is traversed. Since the travel speed is only modified in some of the time periods of these arcs, these arcs still have time-dependent travel times. The time-dependent travel times on all other arcs are not modified. Finally, when violations occur due to the time window restrictions, the time windows are altered to match the optimal time-independent solution. In this way, it is artificially ensured that the time-independent optimal solution is also an optimal solution to its time-dependent variant.

When comparing the results, displayed in Table 1, obtained by our approach on these 36 instances with the known optimal solution, an average gap of 0.2% is obtained, with an average computation time of 0.3 seconds. The maximal gap is 4.1% and the maximal computation time is 0.7 seconds. Furthermore, we conclude that the strength of this algorithm lies in the interaction effect of the repeated insert and replace local search procedures.

4 Conclusion

This paper presents the time-dependent orienteering problem with time windows, a relatively unstudied extension to the orienteering problem where time windows are considered together with travel times which depend on the departure time. In the considered tourist and logistical applications, this modification models multi-modal transport functionality, as well as congestion troubled vehicle routing planning. Most of these applications require solutions within a short amount of computation time.

The proposed algorithm obtains promising results on realistic time-dependent test instances, based on the open source database OpenStreetMap. On average, the algo-

Table 1 Results for the TD-OPTW

Amount vertices	best % gap	avg % gap	worst % gap	CPU s
20	0.0	0.0	0.0	0.1
50	0.0	0.2	0.8	0.3
100	0.0	0.1	0.5	0.4
250	0.9	1.0	1.1	0.6
overall maximum	3.6	4.1	6.4	0.7
overall average	0.1	0.2	0.5	0.3
% optimal	96.8	83.9	83.9	

rithm obtains solutions with an score gap of 0.2% using 0.3 seconds of computation time. The fast execution time of this algorithm enables some interesting business applications where it is necessary to update routes when new traffic information becomes available and to provide proper guidance to truck drivers or tourists on the road.

Further research will focus on fine-tuning and testing the proposed algorithm on a very large network like the Benelux road network with detailed historic travel time information per road segment. Subsequently, the time-dependent variant of the team orienteering problem (TD-TOPTW) will be researched. This rather interesting extension of the problem allows to optimize the routing of a fleet of vehicles, instead of only one vehicle, which is certainly useful for logistic companies.

References

1. Abbaspour A, Samadzadegan F (2011) Time-dependent personal tour planning and scheduling in metropolises. Expert Systems with Applications 38:12,439–12,452
2. Donati A, Montemanni R, Casagrande N, Rizzoli A, Gambardella L (2008) Time dependent vehicle routing problem with a multi ant colony system. European Journal of Operational Research 185:1174–1191
3. Garcia A, Vansteenwegen P, Arbelaitz O, Souffriau W, Linaza M (2013) Integrating Public Transportation in Personalised Electronic Tourist Guides. Computers & Operations Research 40:758–774
4. Ke L, Archetti C, Feng Z (2008) Ants can solve the team orienteering problem. Computers and Industrial Engineering 54:648–665
5. Li J, Wu Q, Li X, Zhu D (2010) Study on the time-dependent orienteering problem. In: International Conference on E-Product E-Service and E-Entertainment (ICEEE)
6. Schilde M, Doerner K, Hartl R, Kiechle G (2009) Metaheuristics for the biobjective orienteering problem. Swarm Intelligence 3:179–201
7. Sörensen K, Sevaux M, Schittekat P (2008) Adaptive, self-adaptive and multi-level metaheuristics, Lecture Notes in Economics and Mathematical Systems, vol 136, Springer, London, chap "Multiple neighbourhood search" in commercial VRP packages: evolving towards self-adaptive methods, pp 239–253
8. Vansteenwegen P, Souffriau W, Van Oudheusden D (2011) The orienteering problem: a survey. European Journal of Operational Research 209:1–10

MISTA 2013

Evacuation Scheduling of a Town after a Natural Disaster: Time-indexed Formulations

Deghdak Kaouthar · T'kindt Vincent · Bouquard Jean-Louis

1 Introduction

Evacuation problems have received increasing attention in the last years. The evacuation models proposed in the literature can be classified into two main categories: macroscopic models and microscopic models. In the former, people are considered homogeneously (i.e. only common behaviours are taken into account) and optimizing their evacuation is usually based on the solution of dynamic network flow problems. In general, it is applied to the evacuation of buildings. In microscopic models, we are interested in taking into account movements and interactions between evacuees when computing an evacuation plan: Microscopic models are more complex and harder to solve than macroscopic models. An extensive survey of these models can be found in Hamacher and Tjandra ([1]).

More recent papers have considered mesoscopic models which are between microscopic models and macroscopic models. They usually involve a discrete simulation that tracks the movements of groups of vehicles. Sbayti and Mahmassani ([2]) and Afshar et al ([3]) have used this approach. The objective in [2] is to minimize the network clearance time. They tackle the problem of evacuation between origin nodes and safety destinations. To find the departure time, route and destination for each evacuee, they use an iterative heuristic procedure and a traffic simulator, named DYNASMART-P. The goal of Afshar et al ([3]) is to determine the choice of shelter locations, traffic assignment and routing. They propose a heuristic method to solve this problem. Chien and Korikanthimath ([4]) propose mathematical models for estimating evacuation time and delay. They specify intensity of threat for each level of urgency, which was the most challenging task in their research.

In our study, we consider large-scale evacuations of mid-size towns, due a naturel disaster like an earthquakes, where evacuees have to change their center of life from several days to several months with the eventual goal of returning back to their respective home. We assume that the locations of gathering points (i.e where people are

Deghdak Kaouthar, T'kindt Vincent, Bouquard Jean Louis
Université François–Rabelais Tours, Laboratoire d'Informatique (EA 6300),
Equipe Ordonnancement et Conduite (ERL CNRS 6305),
64 Avenue Jean Portalis, 37200 Tours, France
E-mail: {deghdak, tkindt, bouquard}@univ-tours.fr

evacuated, outside the town), collection points (i.e where people are gathered waiting to be evacuated, inside the town) and the capacities of the transportation network are known. The goal is to define a macroscopic plan of evacuation to transport evacuees from the collection points to the gathering points in the minimal amount of time. The evacuation is done by means of a fleet of homogeneous buses. This problem can be modeled as a cumulative scheduling problem with additional resource constraints: the gathering points are cumulative parallel machines and the jobs to schedule are the persons to evacuate. The additional resources are the buses which are disjunctive resources. One other important feature of our evacuation scheduling problem is that the processing time of the jobs are dependent on the starting time of the job in the schedule. This is a consequence of the evolution of the transportation network through time due to events like earthquake replicates, road repairs,....

In section 2, we define the evacuation scheduling problem into details and we next provide a time-indexed formulation.

2 Problem description and time-indexed formulation

Consider that we have M parallel and cumulative machines, each machine j corresponding to a gathering location denoted by GP_j, with a capacity cap_j expressed as a number of buses that can bring evacuees. N evacuation operations have to be scheduled on the machines, and each operation i is associated with a collection point CP_ℓ. Similarly to the calculation of the machine's capacities, it is assumed that one evacuation operation coressponds to the evacuation of persons by a filled bus. Thus, the set of evacuation operations is deduced from the number of persons to evacuate and the capacity of a bus. It is assumed to be given and let K be the number of buses available. An additional assumption is made on the time horizon since, on the original problem, events that modify the transportation network, may occur: we assume that k such events happen at a known time d_l and can change the value of processing time of operation i on machine j:

$$p_{i,j} = \begin{cases} a_{i,j}^0 & \text{if } t_{ij} \in]0, d_1] \\ a_{i,j}^1 & \text{if } t_{ij} \in]d_1, d_2] \\ \vdots & \\ a_{i,j}^{k-1} & \text{if } t_{ij} \in]d_{k-1}, d_k] \end{cases}$$

where $a_{i,j}^X$ is the processing time (evacuation time) if the operation i starts to be processed on machine j in the x^{th} time interval, i.e it's starting time $t_{ij} \in]d_{x-1}, d_x]$. The number of finite intervals $[d_{l-1}, d_l]$ is determined by a preliminary analysis of the evolution of the transportation network according to the environment conditions. We suppose that the capacity of the reconfigured network is sufficient after the disaster to enable a fluent evacuation. This hypothesis enables to consider only the transport duration (i.e duration of evacuation).

To model this evacuation scheduling problem we have proposed a time-indexed mathematical formulation. Usually, time-indexed formulations on scheduling problems yield to simple and efficient models despite the presence of a pseudo-polynomial number of variables (see [6]). Let us turn to the model for our evacuation scheduling problem in which T is the time horizon. The decision variables are:

$$\forall i = \{1, 2, .., N\}, \forall j = \{1, 2, .., M\}, \forall t = \{0, 1, .., T-1\};$$

$$x_{i,j,t} = \begin{cases} 1 \text{ if a bus starts the evacuation operation } i \text{ on } j \text{ at } [t, t+1[\\ 0 \text{ otherwise.} \end{cases}$$

and C_{max}, the duration of the evacuation schedule.

The formulation is as follows :

$$\min C_{max}. \tag{1}$$

Subject to:

$$C_{max} \geq (t + p_{i,j,t})x_{i,j,t} \qquad \forall i = \{1, .., N\}, \forall j = \{1, .., M\}, \forall t = \{0, .., T-1\} \tag{2}$$

$$\sum_{t=0}^{T-1} \sum_{i=1}^{N} x_{i,j,t} \leq cap_j \qquad \forall j = \{1, 2, .., M\} \tag{3}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{(t' \in 0..t \text{ et } p_{i,j,t'} + t' > t)} x_{i,j,t'} \leq K \qquad \forall t = \{0, 1, .., T-1\} \tag{4}$$

$$\sum_{t=0}^{T-1} \sum_{j=1}^{M} x_{i,j,t} = 1 \qquad \forall i = \{1, 2, .., N\} \tag{5}$$

Constraints (2) require C_{max} to be greater than or equal to the completion time of any operation, which is then minimized by the objective function (1). Constraints (3) are the gathering point capacity constraints: we can not exceed the capacity of gathering points. Constraints (4) are the bus capacity constraints ; we can not exceed the number of busses we have. Constraints (5) ensure that each operation must be processed once and only once.

The solution of the time-indexed formulation is done by use of IBM CPLEX 12.2. A first evaluation of our model on instances with two parallel machines and a number of operations less than sixty leads to the conclusion that CPLEX is able to solve them to optimality in less than 322 seconds on the average.

3 Conclusions

In this abstract, we have presented one mathematical formulation. During the conference, we will present another time-indexed mathematical formulation with other evaluations. We will also give a comparison between the two formulations in terms of the execution time, the number of explored nodes,....

As a future research line, we can outline the need for heuristic procedures to solve real-size instances. First, we believe the problem exhibits a structure close to that an assignement problem which could leads to efficient constructive heuristics. Besides, the time-indexed formulation proposed for the evacuation problem can be applied to construct a matheuristic for the solution of the scheduling problem. Which means that we look for an amelioration of an initial heuristic solution of the scheduling problem by computing its neighberhood using the mathematical solver.

Acknowledgements This research has been supported by ANR - 11-SECU-002-01, project DSS_EVAC_LOGISTIQUE (CSOSG 2011)

References

1. H.W. Hamacher and S.A Tjandra. Mathematical modeling of evacuation problems: State of the art.Pedestrian and Evacuation Dynamics, European Journal Of Operational Research,227-266 (2001).
2. Hayssam Sbayti and Hani S. Mahmassani, Optimal Scheduling of Evacuation Operations,Journal of the Transportation Reasearch Board, 1964, 238-246 (2006)
3. Afshar and Haghani, Heuristic Framework for Optimizing Hurricane Evacuation Operations, Transportation Research Record, 2089, 9-17 (2008).
4. Chien and Korikanthimath, Analysis and Modeling of Simultaneous and Staged Emergency Evacuations, Journal of Transportation Engineering,133(3), 190-197(2007).
5. S. Gawiejnowicz. Time-dependent scheduling, 377, Springer,Berlin (2008)
6. Berghman, L, Leus, R. and Spieksma, Optimal solutions for a dock assignment problem with trailer transportation, Annals of Operations Research, DOI: 10.1007/s10479-011-0971-7, F.C.R. (2011)

MISTA 2013

The Impact of Problem Characteristics on the Robustness of a Personnel Schedule: An Empirical Investigation

Jonas Ingels • Broos Maenhout

1 Introduction

Personnel scheduling is an important aspect in every organization. The goal is to meet the customer demands as much as possible while offering an attractive roster to employees [1]. The latter is usually determined from an employee's point of view [2]. The first objective is, however, less clear. Meeting the customer demands as much as possible indicates that the organization should be able to cope with unexpected events at the time of execution, due to variability in supply and demand. Both these objectives have to be met while minimizing the total cost for the organization.

Different circumstances may arise that render it impossible for organizations to sustain their preferred service level, i.e., meeting customer demands. Examples are machine breakdowns, employee absence, etc. These disruptions force organizations to intervene for instance by using reserve employees, applying overtime, consciously not meeting customer demand, and assigning employees who were scheduled to have a day off [3]. Note that changes to the personnel roster can lead to violations of personnel preferences and to significant differences between planned and actual costs. As a consequence, robustness becomes more and more important.

In this paper, we examine a reactive scheduling approach, where we try to derive some general guidelines to construct a predictive robust schedule to deal with uncertainty. We evaluate different parameters and their respective impact on the robustness of personnel schedules. Their impact will be studied under different circumstances, which will be simulated using discrete event simulation.

Jonas Ingels
Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2,
9000 Gent (Belgium)
E-mail: Jonas.Ingels@Ugent.be

Broos Maenhout
Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2,
9000 Gent (Belgium)
E-mail: Broos.Maenhout@Ugent.be

2 Problem Description

In our research, we consider the tactical phase of personnel planning, where a personnel roster is constructed for the next month. We assume that, in this phase, all input is known and deterministic. Given the estimated daily shift requirements, individual personnel schedules are constructed for the different employees. However, on the operational level, every organization is subject to different forms of uncertainty, leading to the importance of robust personnel scheduling during the tactical phase (cf. section 1). [4] use three categories of uncertainty in their classification, which are uncertainty of demand, uncertainty of arrival, and uncertainty of capacity.

In this paper, we study a general personnel task scheduling problem. The personnel information, task characteristics, objectives, and constraints are general and common in literature [5]. In our test design, we construct personnel rosters for up to 20 employees and a planning period of 28 days. We consider a set of constraints, consisting of those constraints that occur frequently in literature and that can be classified into coverage constraints, time related constraints, and work regulations [5]. The coverage constraints will consist of minimum and preferred requirements and, for the time related constraints, we define minima and maxima, in accordance with [6].

3 Methodology

The tactical personnel schedule is constructed using a branch-and-price procedure. These resulting schedules are then tested using discrete event simulation. In this operational phase, we simulate variability in supply and demand using different stochastic distributions. Next, we evaluate the outcome and, if necessary, recourse actions will be taken, based on a re-optimization of the model using integer programming optimization [7].

4 Computational Experiments

The experiments consist of two phases. In the first phase, the above-mentioned uncertainty classes are considered separately. In the second phase, the integration and interaction of these classes are handled. Within each of these two phases, we constructed a test design where the problem definition evolves from simple to more complex as a result of the gradual inclusion of problem characteristics, i.e., additional objectives and constraints. Additionally, changes in the constraint parameter values are tested. It is, for example, possible to change the minimum and maximum required number of consecutive working days or to include penalties for not meeting the preferred requirements instead of the minimum requirements. Furthermore, the impact of different parameters on the robustness is examined in each phase. One of these parameters is the competence level of employees. This parameter determines whether it is possible to downgrade, and thus to achieve a higher flexibility in rescheduling, or not.

In both phases, we test the robustness by checking what happens if certain changes occur [8]. The robustness of the schedule is then determined by comparing the initial schedule with the adjusted schedule. For every disruption, we count the number of changes that had to be made and this number will then be used as a measure for the robustness of the schedule.

References

1. 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.
2. M. Vanhoucke and B. Maenhout. On the characterization and generation of nurse scheduling problem instances. European Journal of Operational Research, 196:457–467, 2009.
3. J. Shebalov and D. Klabjan. Robust Airline Crew Pairing: Move-up Crews. Transportation Science, 40:300–312, 2006.
4. J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, and L. De Boeck. Personnel scheduling: A literature review. European Journal of Operational Research, 226:367–385, 2013.
5. E.K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. Journal of Scheduling, 7:441–499, 2004.
6. M. Vanhoucke and B. Maenhout, NSPLib – A nurse scheduling problem library: A tool to evaluate (meta-)heuristic procedures. Operational Research for Health Policy: Making Better Decisions, 151-165, 2007
7. J.F. Bard and H.W. Purnomo. Hospital-wide reactive scheduling of nurses with preference considerations. IIE Transactions, 37:589–608, 2005.
8. K.A. Dowsland and J.M. Thompson, Solving a nurse scheduling problem with knapsack, networks and tabu search, Journal of the Operational Research Society, 51:825-833, 2000.

MISTA 2013

New MILP Formulations for Minimizing the Number of Tardy Jobs on a Single Batch Machine

Stéphane Dauzère-Pérès · Lars Mönch

1 Introduction and Problem

Scheduling problems for batch processing machines are important in semiconductor manufacturing because the processing times of the batching operations are much longer than those for operations on non-batching machines [6]. As a result, batching decisions have an impact on the dynamic behavior of the entire wafer fabrication facility (wafer fab). A batch is defined as a group of jobs that are processed simultaneously on one machine.

In this paper, we consider a single-machine batching problem. Each job j belongs to an incompatible job family denoted by $s(j)$. Each single job j has a due date d_j and a weight w_j. The goal is to minimize the weighted number of tardy jobs, defined as $\sum w_j U_j$, where $U_j = 1$ if $C_j > d_j$ for the completion time C_j of job j in the schedule, and $U_j = 0$ otherwise. The following assumptions are made:

1. Jobs of the same family s, $s = 1, \ldots, f$ have an identical processing time p_s. Due to the different chemical nature of the processes, only jobs of the same family can be batched together. This is typical for diffusion furnaces in wafer fabs.
2. The n jobs are available at time $t = 0$, i.e., we have $r_j = 0$, where r_j is the ready time of job j.
3. Once a batch processing machine is started, it cannot be interrupted. No preemption of the processing of a batch is allowed.

At maximum B jobs can be processed simultaneously on the machine, i.e. B is the maximum batch size. Now the problem can be represented as

$$1|p - batch, incompatible| \sum w_j U_j \tag{1}$$

Stéphane Dauzère-Pérès
Department of Manufacturing Sciences and Logistics, Centre Microélectronique de Provence - Site Georges Charpak, Ecole des Mines de Saint-Etienne
E-mail: Dauzere-Peres@emse.fr

Lars Mönch
Department of Mathematics and Computer Science
University of Hagen
E-mail: Lars.Moench@fernuni-hagen.de

using the $\alpha|\beta|\gamma$ notation (cf. [3]). We consider the criterion $\sum U_j$, i.e. $w_j = 1$, $j = 1, \ldots, n$, in the unweighted case. It is shown in [5] that problem (1) is NP-hard even in the unweighted case. Dynamic programs are formulated for the unweighted and weighted case, respectively, in [4,5]. But these formulations are not tractable from a computational point of view if $f > 1$ is assumed. Therefore, we proposed two Mixed Integer Linear Programming (MILP) formulations based on positional variables and a random key genetic algorithm in [2]. The two MILP models $LPB1$ and $LPB2$ allow problem instances up to 72 jobs to be solved to optimality in a reasonable amount of time.

In this paper, we propose two new MILP formulations that significantly outperform the MILP formulations presented in [2], i.e. problem instances up to 144 jobs can be solved to optimality in almost all situations. The new formulations take into account insights in the problem structure in a better way than those in [2]. They were motivated by MILP formulations for the problem $1|r_j|\sum w_j U_j$ that are based on the concept of a master sequence [1]. The main idea is that we can consider the smallest due date of the jobs that form a batch as a due date of the batch. This allows an EDD-sorting of the jobs in a preprocessing step.

2 MILP Formulation for the Unweighted Case

In $LPB3$, each position will be occupied by at most one batch. Let us assume that jobs are ordered in the non-decreasing order of their due dates, i.e. $i < j$ implies $d_i \leq d_j$. The binary variable v_k, $k = 1, \ldots, n$, specifies whether a batch of family $s(k)$ is generated by job k in position k, i.e. k is the job with the smallest index in the batch, or not. From Uzsoy [7], we know that, for regular objective functions, there is an optimal schedule where all batches, except possibly the last scheduled batch of each job family, are fully loaded. Moreover, in the unweighted case, we know there is an optimal solution such that, if job i is on time, then all jobs j such that $j > i$ and $s(j) = s(i)$ are also on time (see Proposition 4 in [2]). Hence, if a batch b is generated by job i, there is an optimal solution in which batch b contains at most B jobs j of family $s(i)$ that follow i. Let jb_k^j be a parameter which is equal to 1 if job j is in the batch generated by job k. By definition, $\sum_{j=k}^{n} jb_k^j \leq B$. The parameters jb_k^j are computed in a procedure that counts the number of jobs of family $s(k)$ after k. Model $LPB3$ is as follows:

$$\min \sum_{j=1}^{n} U_j \tag{2}$$

subject to

$$C_1 - p_{s(1)} v_1 \geq 0, \tag{3}$$

$$C_k - C_{k-1} - p_{s(k)} v_k \geq 0, \quad k = 2, \ldots, n, \tag{4}$$

$$C_k \leq d_k, \quad k = 1, \ldots, n, \tag{5}$$

$$\sum_{k=1}^{n} jb_k^j v_k + U_j = 1, \quad j = 1, \ldots, n, \tag{6}$$

$$v_k \in \{0,1\}, \quad k = 1, \ldots, n, \tag{7}$$

$$U_j \in \{0,1\}, \quad j = 1, \ldots, n, \tag{8}$$

$$C_k \geq 0, \quad k = 1, \ldots, n. \tag{9}$$

Constraints (3) and (4) relate the completion time of the batch at position k to the completion time of the batch at position $k-1$. Constraints (5) ensure that the batch

generated by job k is completed before its due date. Constraints (6) specify that job j is on time, i.e. a batch that contains j is started, or is tardy. The remaining constraints ensure that the decision variables are binary or non-negative.

3 MILP Formulation for the Weighted Case

In the weighted case, there is not always an optimal solution in which, if job i is on time, then all jobs j such that $j > i$ and $s(j) = s(i)$ are also on time. Hence any job after i of the same family can be in the batch generated by i. This is why we need an additional variable u_k^j which is equal to 1 if job j is in the batch generated by job k and zero otherwise. Note that there is an optimal solution in which u_k^j is equal to 1 only if $s(j) = s(k)$ and $j \geq k$. Model $LPB4$ is as follows:

$$\min \sum_{j=1}^{n} w_j U_j \tag{10}$$

subject to

$$C_1 - p_{s(1)} v_1 \geq 0, \tag{11}$$

$$C_k - C_{k-1} - p_{s(k)} v_k \geq 0, \quad k = 2, \ldots, n, \tag{12}$$

$$C_k \leq d_k, \; k = 1, \ldots, n, \tag{13}$$

$$\sum_{j=k;\; s(k)=s(j)}^{n} u_k^j \leq B, \; k = 1, \ldots, n, \tag{14}$$

$$\sum_{k=1;\; s(k)=s(j)}^{j} u_k^j + U_j = 1, \quad j = 1, \ldots, n, \tag{15}$$

$$v_k \in \{0, 1\}, \quad k = 1, \ldots, n, \tag{16}$$

$$u_k^j \in \{0, 1\}, \quad k = 1, \ldots, n, j = 1, \ldots, n, \tag{17}$$

$$U_j \in \{0, 1\}, \quad j = 1, \ldots, n, \tag{18}$$

$$C_k \geq 0, \quad k = 1, \ldots, n. \tag{19}$$

Constraints (11) - (13) are similar to Constraints (3) - (5) from $LPB3$. The maximum batch size is ensured by Constraints (14). Constraints (15) are again similar to Constraints (6) in $LBP3$. The same is true for the remaining constraints.

4 Computational Experiments

We use the same problem instances as in [2]. Small- and medium-size problem instances up to 72 jobs and large-size problem instances with 144 jobs are considered. In addition, we also solve 72 huge-size problem instances with 256 jobs. The maximum batch size, the number of families, the due date tightness and range are factors of the experimental design. All the computational experiments were conducted on a computer with an Intel Xeon 2.67 GHz processor and 3.48 GB RAM using IBM ILOG CPLEX 12.5. The maximum computing time per instance is varied between 60, 600, and 3600 seconds.

We show the results for the unweighted and weighted cases in Table 1. Each entry in the table contains the number of problem instances where optimality is not proven and the average MIP gap for all instances, respectively. We also show the corresponding results for the MILP model $LPB1$ from [2] that contains a big-M coefficient. $LPB1$ is

Table 1 Results for unweighted and weighted problem instances

	MILP model	Time (seconds)	Instances		
			Small	Large	Huge
Unweighted	$LPB3$	60	0	0	0
	$LPB1$ from [2]	600	3 (1.7%)	38 (9.6%)	52 (15.6%)
Weighted	$LPB4$	60	1 (0.4%)	19 (1.6%)	38 (3.0%)
		600	1 (0.4%)	15 (0.7%)	35 (1.0%)
		3600	1 (0.3%)	12 (0.4%)	35 (0.9%)
	$LPB1$ from [2]	600	8 (1.1%)	47 (13.8%)	58 (20.0%)

selected for comparison because the difference between $LPB1$ and $LPB2$ is rather small and $LPB1$ often performs slightly better. The new formulation clearly outperforms the old one in the unweighted and weighted cases.

A maximum computing time of 60 seconds is enough to solve all problem instances, even the huge-size ones, to optimality in the unweighted case. The weighted problem instances are harder to solve and $LPB4$ contains an additional binary variable. However, the number of problem instances without not proven optimal is still small. The MIP gap is often close to zero for 600 seconds and 3600 seconds of maximum computing time. This indicates that we have high-quality solutions.

These encouraging results illustrate that MILP formulations have a strong impact on the solution quality when using a standard solver. It is worth to exploit the structure of optimal solutions to improve MILP formulations.

References

1. Dauzére-Pèrés, S., Sevaux, M. Using Lagrangean Relaxation to Minimize the Weighted Number of Late Jobs on a Single Machine. *Naval Research Logistics*, 50(3), 273-288 (2003)
2. Dauzére-Pèrés, S., Mönch, L. Scheduling Jobs on a Single Batch Processing Machine with Incompatible Job Families and Weighted Number of Tardy Jobs Objective. *Computers and Operations Research*, 40, 1224-1233 (2013)
3. Graham, R. L., Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of Discrete Mathematics*, 5, 287-326 (1979)
4. Jolai, F. Minimizing Number of Tardy Jobs on a Batch Processing Machine with Incompatible Job Families. *European Journal of Operational Research*, 162, 184-190 (2005)
5. Liu, L., Zhang, F. Minimizing Number of Tardy Jobs on a Batch Processing Machine with Incompatible Job Families. *Proceedings of the 2008 ISECS International Colloquium on Computing, Communication, Control, and Management*, 277-280 (2008)
6. Mönch, L., Fowler, J. W., Dauzére-Pèrés, S., Mason, S. J., Rose, O. Scheduling Semiconductor Manufacturing Operations: Problems, Solution Techniques,and Future Challenges. *Journal of Scheduling*, 14(6), 583-595 (2011)
7. Uzsoy, R. Scheduling Batch Processing Machines with Incompatible Job Families. *International Journal of Production Research*, 33(10), 2685-2708 (1995)

MISTA 2013

An initial study of a novel Step Counting Hill Climbing heuristic applied to timetabling problems

Yuri Bykov • Sanja Petrovic

1 Introduction

A local search algorithm known as Late Acceptance Hill Climbing (LAHC) was proposed by Burke and Bykov in [2]. It is an extension of the Hill Climbing (HC) algorithm where a candidate solution is accepted when it is better than a solution, which was the current several iterations ago. The method has attracted a considerable attention from the research community because it is powerful enough, easy to implement and has distinct properties, which make it effective in practical applications. A number of researches from different institutions have started their own studies of this method, which resulted in published papers (e.g. [5]). Also, LAHC has got the top scores in two optimisation competitions. In this study we propose a new extension of HC, which keeps all properties of LAHC but it is even simpler and has certain additional advantages.

2 The Step Counting Hill Climbing

The main concept of our proposed algorithm is based on the idea of prolongation of the Hill Climbing search procedure. In other words, if we temporarily allow HC to accept worsening moves then this could increase the search time, but it is to be expected that the results might be better than the HC ones. Although a lot of variants of the implementation of this idea are possible, the proposed one is based on the following interpretation of the HC acceptance condition: in HC only solutions that are not worse than the current one are accepted, so we can regard that as the current cost serves as an (upper) cost bound (B_c) for the next iteration (step). Thus, the search can be prolonged when *the given cost bound is kept not just for one, but for a number of consequent steps* (during which the worsening moves can be accepted).

This idea can be implemented using a simple step counting mechanism. Here, after each updating of B_c (making it equal to the current cost value), the algorithm counts consequent steps (n_c) and when their number exceeds a given counter limit (L_c) the cost bound is updated again (and so on). Hence, we have named this algorithm as the Step Counting Hill Climbing (SCHC). Besides the new acceptance condition, the method follows the basic layout of the local search algorithms: starting from a random initial solution it iteratively accepts or rejects candidates until a stopping condition is met.

The initialization of SCHC is also straightforward. We suggest assigning the initial value of B_c to the initial cost function. The only parameter the user should specify is L_c. It should be also noted that this algorithm operates with four types of moves (steps): (1) the moves where candidate cost is better than the current one (improving moves), (2) the moves where candidate cost is equal to the current one, (3) worsening accepted moves and (4) worsening not accepted

Yuri Bykov
Nottingham University Business School, Jubilee Campus, Wollaton Road, NG8 1BB, Nottingham, United Kingdom
E-mail: yuri.bykov@nottingham.ac.uk

Sanja Petrovic
Nottingham University Business School, Jubilee Campus, Wollaton Road, NG8 1BB, Nottingham, United Kingdom
E-mail: sanja.petrovic@nottingham.ac.uk

moves. Correspondingly, we can count the moves of just a single type or some combination of them, for example, to count the improving and equal moves (type 1 + type 2). As four move types can compose in total 15 different combinations (combination which consists of no moves is of no interest), then, (depending on the types, which are actually counted) SCHC can be implemented in 15 variants. We have tested all these variants and our experiments have shown that their best performance is quite problem-dependent. However, on average (over all our tested instances) the most advantageous strategy is to count just all accepted moves (type 1 + type 2 + type 3). Therefore, namely this variant is studied below. In addition, a stronger performance was shown when the algorithm accepts the candidates with the cost function namely *better* (not *better or equal*) than B_c. However, using this acceptance rule the algorithm could miss some good solutions, hence we also suggest (in analogy to other local search methods, such as Simulated Annealing (SA) or LAHC) to combine the step counting acceptance rule with the greedy rule. The pseudocode of the final version of SCHC heuristic (for minimization problems) can be outlined in Figure 1.

```
Produce an initial solution s
Calculate an initial cost function C(s)
Initial cost bound Bc:=C(s)
Initial counter nc:=0
Specify Lc
Do until a chosen stopping condition
   Construct a candidate solution s*
   Calculate a candidate cost function C(s*)
   If C(s*)<Bc or C(s*)≤C(s)
      Then accept the candidate s:=s*;
           increment the counter nc:=nc+1
      Else reject the candidate s:=s
   If nc ≥ Lc
      Then update the bound Bc:=C(s);
           reset the counter nc:=0
```

Figure 1: The pseudocode of the Step Counting Hill Climbing (SCHC)

Our initial tests with SCHC have revealed that the properties of this algorithm are very similar to LAHC discussed in [3]:

- The algorithm requires only one parameter L_c ("counter limit") to be set up.
- When $L_c \leq 1$ the search turns into greedy HC. Hence, SCHC obtains its own properties with values of $L_c > 1$. With the increase of L_c the search time is also increased which (for many problems) leads to better final results.
- The total CPU time of the search is approximately proportional to L_c, however the coefficient of proportionality is different for different problems and is generally larger for larger problems.
- The proposed variant of the prolongation of HC does not affect its scale independence; i.e. SCHC works well on non-linearly rescaled problems where many known methods (for example, SA) fail to produce good results (see an example of such problem in [3]).

Note, although the proposed algorithm is very close to LAHC, it does not operate with the list of previous cost functions. Instead, SCHC employs a simple counting mechanism, where the counter limit L_c can be easily adjusted at any time. This could be seen as an additional advantage of SCHC, in contrast to LAHC, where the dynamic variation of the length of the list is a more complicated procedure. The simplicity of the dynamic adjustment of the search process suggests that SCHC can have a considerable number of variants and modifications. In particular, SCHC can serve as a good platform for experiments with different search patterns, for investigating the search behavior and for developing self-adaptive heuristics.

3 Experiments

We compare the performance of SCHC with two most related local search algorithms: Simulated Annealing and LAHC. To make the comparison fair, all experiments were done using the same software described in [3] while adapting the acceptance condition for each particular heuristic. The software was developed in Delphi 2007 and run on PC Intel Core i7-3820 3.6 GHz, 32 GB RAM under OS Windows 7 64 bit. To ensure the adequate performance of SA we use general suggestions from the literature for its parameterization. Thus, we employ geometric cooling schedule where the initial temperature is set up in such a way that at the starting point the algorithm accepts 85% of non-improving moves. The experiments were done with 12 university exam timetabling benchmark instances, introduced in the examination track of the ITC2007 timetabling competition. Their description is given in [4] and on the competition web site: http://www.cs.qub.ac.uk/itc2007/. All compared algorithms allow an approximate setting of the processing time (see [1]), so we compare the results of three methods produced in the same CPU time. Table 1 presents the results of experiments with the run time set to 120 seconds. These are the average results collected over 50 runs. The best results over 3 competitor heuristics are highlighted in bold.

Table 1 Comparison of SA, LAHC and SCHC with the run time set to 120 seconds.

Dataset	SA	LAHC	SCHC
Exam_1	4194	4034	**4008**
Exam_2	**396**	406	404
Exam_3	8391	8031	**8012**
Exam_4	12912	**12817**	13312
Exam_5	2620	2603	**2582**
Exam_6	25470	**25421**	25448
Exam_7	4063	3930	**3893**
Exam_8	7126	**6925**	6944
Exam_9	965	963	**949**
Exam_10	13008	13001	**12985**
Exam_11	25686	**24916**	25194
Exam_12	**5166**	5187	5181

This table demonstrates that although SCHC is a very simple heuristic, its performance is sufficiently strong. It outperforms both competitor methods on 6 out of 12 benchmark problems. In particular, SCHC performs better than SA on 9 problems, better than LAHC on 8 problems and only on one problem the SCHC is inferior to the both competitor methods. In contrast, SA has the worst performance over three methods, namely on 9 problems its results are worse than either of LAHC or SCHC and only on two instances SA performs well. We believe that the demonstrated strong performance of SCHC together with its unique properties could motivate its further studies and practical applications.

References

1. Burke, E. K., Y. Bykov, J. Newall, S. Petrovic, A time-predefined local search approach to exam timetabling problems, IIE Transactions 36(6), 509-528, (2004).
2. Burke E. K., Y. Bykov, A late acceptance strategy in Hill-Climbing for exam timetabling problems, (extended abstract), in Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT2008), Montreal, Canada, (2008).
3. Burke E. K., Y. Bykov, The Late Acceptance Hill Climbing heuristic. Technical report CSM-192 Computing Science and Mathematics, University of Stirling, UK, (2012).
4. McCollum, B., A. Schaerf, B. Paechter, P. J. McMullan, R. Lewis, A. J. Parkes, L. Di Gaspero, E. K. Burke, R. Qu, Setting the research agenda in automated timetabling: The second international timetabling competition, INFORMS Journal on Computing, 22, 120-130, (2010).
5. Verstichel J., G. Vanden Berghe, A late acceptance algorithm for the lock scheduling problem, Logistic Management, 2009 (5), 457-478, (2009).

MISTA 2013

Use Matrixes Of Weighing for the scheduling in a sports Fantasy.

Javier J. Hernández G. • María J. García G. • José G. Hernández R.

Abstract The sports business in the last years has generated a new source of money; this is Fantasy Sports or Fantasy Leagues. Across a fantasy, the participants turn into owners and managers or coaches of their teams of fantasy, developing with it their capacity for the decision making. Many of them involve a decision making in the long term and a constant planning of the weekly activities of the team. In the case of soccer, the big clubs and with them many of the best players, usually have multiple commitments in the same week. So for these weekly schedules, that involve the respective fantasy leagues, it was thought one of the multicriteria models simpler to implement, the Matrixes Of Weighing. From the above, the objective of this work is to use the Matrixes Of Weighing to carry the selection of the players of its squad that will place as assets, a fantasy participant, in each of the weeks on the league.

Keywords: Fantasy Sports, Matrixes Of Weighing, Scheduling, Soccer, Decisions.

1 Introduction

The sports business has captured a lot of fans through fantasy leagues. And as stated Massari (2006) [11], the participation in the games of fantasy is an industry of multimillionaires proportions and is ratify by Burke (2008) [1], when says that the business of fantasy sports is a multimillion dollar industry. These leagues exist practically for all the sports and they have been expanded throughout the world. One of the sports where they are many participants in many fantastic leagues is the sport of soccer. In general these leagues are established for country, but also can be simultaneously engaged, in the case of Europe, in the league that covers the UEFA Champions League, where the better teams participate of the different European leagues and the UEFA Europe League or even the World Cup that is celebrate every four years. To participate in a fantastic league relating to the soccer, as in almost all the sports, the first thing that it is necessary to do is to define a squad of players. To

Javier J. Hernández G.
Gerencia de Investigación, Minimax Consultores, Bristol Inglaterra.
E-mail: minimax@cantv.net

María J. García G.
Gerencia General, Minimax Consultores, Caracas Venezuela.
E-mail: Marimininagarcia@yahoo.com

José G. Hernández R.
Departamento de Gestión de la tecnología, Universidad Metropolitana, Caracas.
E-mail: jhernandez@unimet.edu.ve

set this squad there is a series of requirements, as amount of money invested, number of players in each position, number of players of each team. But is not enough to have a good squad to win a fantastic league, but there is also necessary to make the most out of the best possible players.

Week to week they should be selection of players in the squad, the ones that are going to be assets, that is to say those who theoretically will be defending the colors of the team of the participant in the fantasy and that in the case of soccer there would be 11 players of different positions. To know that player must be activated there is a series of criteria that must be taken into account, for which it is possible to visualize this decision, like a multicriteria problem. One technique Multicriteria simplest model and implement are the Matrixes Of Weighing (MOW) (García and Hernández, 2011 [6]; Hernández, García and Hernández, 2011 [9]). The possibility of using a MOW to set the weekly scheduling of a participant in a fantasy soccer, becomes the objective of this work: to use the Matrixes Of Weighing to carry the selection of the players of its squad that a participant in fantasy will place as assets, each of the weeks of the league.

This general objective involves three specific objectives: To present the MOW, Analyze the scheduling to be established for every week that lasts the fantasy and to create and to present the model who allows handling this situation.

To achieve this overall objective and specific objectives will follow the Integrated-Adaptable Methodology for the development of Decision Support System (IAMDSS, in Spanish, Metodología Integradora-Adaptable para desarrollar Sistemas de Apoyo a las Decisiones [MIASAD]) (García, Hernández and Hernández, 2011 [7]). It will take advantage of the condition of adaptable of the methodology, to take, as they have done in other works (García and Hernández, 2011 [6]; Hernández, García and Hernández, 2011 [9]; 2012 [10]), only some of the essential steps of the same one:

a) To define the problem that, as is indicated in the objectives, is establishing the scheduling of the players, that will be placed as assets in a fantasy, departing from a MOW,

b) To prepare the first prototype, where there identified the users of the final product, that is to say the principal readership of the article, who are all interested in the application of quantitative tools in the playful that will join those soccer fans typically involved in a fantasy. Also, there was established the structure of the article, which in addition to the introduction will consist of three central chapters, in the last, and main one will show the steps to choose the players in the squad that will be activated each week,

c) Obtaining data, in this case on soccer, fantasy, scheduling and MOW,

d) Establishing alternatives, that would be the different options that it could be used the MOW to choose of a squad of players of a fantasy, the ones that will be start a week,

e) Evaluate alternatives, according to the performance expected of the activated players and of the facility of implementing the select strategy,

f) Selecting the best alternative, as product of previous evaluation process, and based on the secondary objectives, tacit or explicit, being considered,

g) Implementing the best alternative, illustrating through a simple example that the chosen model permits to select the best group of players to activate and

h) Establishing controls, Establishing the mechanisms, probably indicators, that permit to recognize if the alternative selected, continues being valid in the course of the time.

With regard to limitations and reaches, in this work will carry out no study of field, although the majority of the information to use are real, these ones do not respond to a participation in a actual fantasy per said, likewise there will not be a monitoring every week, but the emphasis will be on how to use the MOW to select players to start. Equally although following a specific fantasy, the study will be as general as possible, to be able to be adapted to the rules of other fantasy.

2 The Soccer fantasy

In the book of Pillay, Tomlinson and Bass (2009) [13], it is possible to see, although only through the mega-events, that the sports business continues being a business in growth and by extension can be affirmed that some countries, despite being in crisis, one of the business that have continue growing is the sports. But the sports now, not only they play in the pitches, but a series of followers of the sports has arisen that feel athletes and they turn into proprietors, managers and coach of his teams, across the fantasy.

"A fantasy sport is a game in which players (i.e., public individuals) compete against one another by maintaining rosters of actual professional athletes and using the real game statistics of those athletes to score points" (Massari, 2006, 444-445) [11].

The origin of the fantasy, is quite old, Oates (2009) [12] indicates that fantasy football was a diversion between journalists and publicists in 60s and Burke (2008) [1] note that the fantasy Baseball is attributed him to Daniel Okrent, journalist of the Sports Illustrated, through the league called "La Rotisserie" in the years 80s.

The impact of fantasy has come to other countries and many sports. Particularly this work will be centered in soccer. The importance of these fantasy soccer is such that they are being used for teaching mathematics, especially to the young people (Flockhart, 2007) [5], in particular if it is possible to affirm that the soccer is not one of the most popular sports in USA (Taylor, Funk and Craighill, 2006) [16], contrary to what happens in the rest of the world. But for teaching and developing the capacity of decision making, soccer presents the challenge that is not a sport that facilitates the measure of the performance of his players (Correll, 2012) [2].

While one of the first decisions in a fantasy, in particular a fantasy soccer, is to conform the squad of players, here the interest will be centered in how utilize the players that integrate this squad in each one of the weeks of participation. Although the model will try to build as general as possible, will be used La Liga (Spanish First Division of soccer) to illustrate how the scheduling will be done with the players in the squad that will be starting every week.

2.1 Fantasy based on La Liga

Every fantasy has some conditions that regulate it and for this work they will follow established in the award "Desafío la liga", without doing of them greater comments and recommended to revise its rules in ESPN (2012) [4], of them, to facilitate the understanding of the model only they will be indicated: They will be used up to 15 players, there cannot be had any more than three players of the same team and regarding positions will not be more than 2 goalkeepers, 5 defenders, 5 midfielders and 3 forwards.

One of the rules has to do with the formations in the field and for it establishes: Each participant should select, of his 15 players, 11 starters for each week of the fantasy. Among those 11 players can only have one goalkeeper (GK) and for the rest of the players some of these formations, orderly: Defender (Df) – Midfielders (Mi) – Forward (Fo): 4-4-2 (for default); 4-3-3; 3-4-3; 3-5-2; 5-3-2; 4-5-1 and 5-4-1.

The participant in the fantasy will obtain or lose points thanks to his eleven active players, considering among other aspects: Matches = 1; 60 Minutes of play = 2; Goal of Df or GK = 6; Goal of Mi = 5; Goal of Fo = 4; Assistance = 3; Clean chart for Df or GK (minimum 60 minutes) = 4; Clean chart for Mi (minimum 60 minutes) = 1; Caught = 1 point by each 3 caught; Penal it Caught = 5; Penal Erroneous = -2; Goals Permitted by GK or Df = -1 point by each 2 goals; Yellow card = -1; Red card = -3; Goal against = -2.

Generally these rules seek to apply a multicriteria approach, whereby further details will be given of the MOW, but previously do some very brief comments on how you would see the scheduling of this fantasy soccer.

2.2 Scheduling on fantasy soccer

It is possible to establish an analogy similar to the one realized for Hamscher et al. (2000) [8] where for them the system of scheduling will be responsible for selecting the best machine to perform a work, in this case is a matter of visualizing the players as the machines and the positions that can perform in the playing field as the tasks. That is why the problem, in a way is summarized to choose, that task will be carried out by each machine. Where for all the tasks (positions) will have more machines (players) available, for that reason, each week, there will be machines that will not be used.

The scheduling to establish, will be centre not on a lot of production, but in who will perform the tasks every week. That is to say is a matter of planning that player will cover each position, in each one of the weeks that is participating in the fantasy. Evidently there is not a scheduling global for all the season, but each week should make a different decision.

To see which player is selected to fill a particular position will be considered a set of criteria: Average performance of players in recent seasons; Performance in the last n games; Team against which is going to face the team each one of the players; Performance, along its career, that has had the player against the team that will face; Current relation of each player with its respective coach and although in many occasions will depend on the previous one and of the health of the player, Possibility that every player has in the next game of being in the field and even it is possible to consider to be an additional criterion, Performance expected of the player that week. All these criteria are analyzed with a little more detail to express the model to use, but previously some comments will be done on the MOW.

3 The Matrixes Of Weighing

Without doubt one of the simplest multicriteria techniques to present and implement are the Matrixes Of Weighing (MOW). As explained (García and Hernández, 2011 [6]; Hernández, García and Hernández, 20111 [9]; 2012 [10]) the MOW are simply an arrangement of rows and columns, which is used to evaluate alternatives through a set of weighted criteria, since it is possible to see in the section 4 in the table 1. Although the MOW are utilized often, they are little indexed in the literature specialized, from there that Hernández, García and Hernández (2012) [10] comment that, in general, of them have only indirect references for jobs with similar or equivalent matrices (Environmental Management Systems, 1997 [3]; Prasad, 1995 [14]; Shen and O'Hare, 2004 [15]).

In the majority of the occasions, in the rows the alternatives are placed and in the columns the criteria, although they do not lose sense if there are invested the positions of alternatives and criteria. Every criterion has a scale of evaluation and to every alternative a value is assigned to it in accordance with the fulfillment that reaches of the respective criterion. In every alternative all these partial evaluations join, to obtain the final value of the alternative (Total Ak).

As multicriteria problems are usually maximization problems, is taken as the best, the highest scoring alternative, but in case of a minimization problem, simply it will be taken like better that of smaller value. The scale of each criterion goes since the initial weight of the criterion j (PCji) to final weight of the criterion j (PCjf), where the subscript j represents the criterion, the subscript i represents the initial value and subscript f the final value and were PCjf should be strictly greater that PCji. The way to define these ranks and the multiplication fixed values that are often used, they allow to differentiate different variants of MOW (García and Hernández, 2011 [6]; Hernández, García and Hernández, 2011 [9]):

- In the first type, for all the criteria, all the PCji they are equals and all the PCjf they are equals, that is to say all the criteria have equal rank.
- In the second type, all the PCji they are used to being equals, although they can be different, but the PCjf, at least one of them is different, for which the criteria have different ranges.

There is a sub-variant for each of two previous types, when it is required to join the ranges set constant value, usually one hundred.

- The third and fourth variant would be equivalent to the first and second respectively, but for each criterion there will be a constant value of multiplication (vj), as it is reflected in (1), were the Pk,j, they are the evaluations of the alternative k for each one of the criteria j, n represents the number of criteria and the values of multiplication (vj) they are usually entire numbers, in general not bigger than ten or twenty.

$$\text{Total } Ak = \sum_{j=1,n} vj * Pk,j \quad (1)$$

- A fifth and sixth variants are equivalent to the third and fourth, but where the values of multiplication are all smaller to 1 and they are required that his sum should be equal to 1, as it is reflected in (2).

$$1 = \sum_{j=1,n} vj \quad (2)$$

Nevertheless in the everyday life he is found different applications of the MOW, strictly not adapted to this structure, but still can be considered MOW. Among these cases of not very formal applications of the MOW many systems of voting are found, particularly they have the following:

1. It distributed a fixed amount of points among the alternatives; in this case they usually use two variants: 1a. With repetition and 1b. Without repetition. In many of their applications some of the alternatives they can remain without receiving points. In the case where all the alternatives should receive points, this modality, it can be seen very similar to the variant one indicated for the type 1 and 2, when it is required that the total sum of the criteria points, was a fixed amount.

2a. They are distributed, among all the alternatives, a quantity of points that go since 1 to m, being m the number of alternatives. To every alternative a different quantity is assigned to it between 1 and m.

2b. Is a variant one of 2a, in this case they are distributed k points, with $k < m$, by which there are alternatives that remain without points. This modality very is seemed to the modality 1a that has just mentioned, when they remain alternatives without receiving points, only this mode is not set the amount of points to be awarded but the number of alternatives that will score.

3. More that a variant is a complement and it is used when there are multiple evaluators. First the set of assessors value all the alternatives across all the criteria, in most cases following a type one MOW. Of all the values obtained, for each criterion the best evaluation reached is taken. This major value for every criterion transforms in a top level (in general 100 is used) and from these new upper values all the alternatives are reappraised, for through this normalization, obtaining more homogeneous values. In practice does not be used to repeating the process of evaluation, but are allocated proportional values and the final sum is redone.

These cases mentioned are usually found in beauty contests and music festivals, in particular to choose the winning song in the Eurovision, each country assigns a fixed quantity of votes, without repetition, to the remaining countries excluding to itself, by which various countries remain without obtaining votes, being thus an example of the modality 2b, before indicated.

3.1 Multiplicative factor on MOW

The fixed multiplicative values (vj), before shown in (1) and (2), not to be confused with the multiplicative factors, that they are an own concept of the Multiattribute Models with multiplicative factors (MMwMf) (García and Hernández, 2011 [6]), that are governed for (3).

$$Pts = \prod_k fgk * (\sum_i \prod_h fih * pci * (\sum_j pajci * vajci)) \quad (3)$$

Where the subscript i represents the criterion and the subscript j the attribute, therefore pci will be the ponderation assigned to the criterion i, pajci will be the ponderation of the attribute j of the criterion i, vajci it will correspond to the value obtained by the attribute j of the criterion i, h will counter the multiplicative factors for each of the criteria i, from there that fih, be the multiplicative factor h of the criterion i, k counts the number of multiplicative factors, that they operate for all the model, which will be called general multiplicative factors, fgk and Pts they will be the total points reached by the alternative in study.

For the MOW, there is not attributes of the criteria, therefore not discuss the criteria multiplicative factors, but only show the general multiplicative factors, from there that (4), becomes a combination of (1) and (3), preserving the nomenclature of (1) and were the fgh will represent the general factors of the MOW, varying from h = 1 to H, H being the total number of general factors.

$$\text{Total } Ak = \prod_{h=1,H} fgh * \sum_{j=1,n} vj * Pk,j \quad (4)$$

Since both for the MMwMf, and for the MOW, the multiplicative factors will be, in general, values between 0 and 1, although is not restrictive, it is recommended that they be not more than three. It is also important to note that these multiplicative factors are not affected by the MOW variant being used at any given time.

3.2 Other aspects of the MOW

By escaping from the reach of this work there are other aspects of the MOW, that do not go away to comment, since they are the use of MOW of multiple layers and MOW and group decision, interested the first case it is recommended to them to consult to Hernández, García and Hernández (2012 [10]) and its sources and for that they wish major information of the second case they can consult to Hernández, García and Hernández (2011) [9] and its sources.

Also do not make comments of the reasons for creating and using multiplicative values (vj) and neither will details the steps to build a MOW, in part by the simple thing, but if there is some interest to do a formal review of the same ones they are in Hernández, García and Hernández (2011) [9] and its sources.

4 The MOW and the scheduling week to week of players

As one commented in the subsection on 2.2, one of the criteria to take into account to decide if a player will participate or not a determined week is the Performance Average Last Season (PALS). For it more than taking into account the performance of the player with regard to its team, it should be taken with regard to the fantasy, that is to say if has generated points according to the rules of fantasy, some of them commented in the subsection 2.1, in every case is recommended to work with the last 45 games of the player, what does not be used to arriving not even to all the play of two seasons. In a similar way, only the performance for the fantasy is seen, for the criterion Performance Average in the last 3 Games (PA3G), and equal with the Life Performance Against the Team to face (LPAT), criterion that its time measures to the criterion Team against which the player is facing. As for Relationship of the Player With its Manager (RPWM) it is a little more speculative and should be based, especially in the press comments.

Of these four criteria by having greater repercussion in direct form will be given it greater weight to PA3G, working this criterion in a scale 1 to 100, second LPAT, that will be measured in a scale 1 to 60, to PALS, since not necessarily it reflects the current performance, will work on a scale 1 to 40 and to RPWM, considering its slightly speculative character, one will work with a scale 1 to 10.

All these criteria will be worked by nine ranks, just as it is shown in the table 2, where every rank represents a minimal and maximum percentage of the points that can be reached in the respective criterion.

The three first criteria PALS, PA3G and LPAT, that come from a numerical value, it can be avoided to do the analysis through the ranks of the table 2, if to the player with greater performance is assigned him the 100% and to all the other players is calculated them a proportional value to this, that will be the procedure to continue in this work. The two last criteria, Possibility of Play This Week (PPTW) and Expected Performance This Week (EPTW) will work as multiplicative factors with values between 0 and 1.

For the first week of the season the statistics of the previous season will be taken and if the player is new in the league will be taken into account its statistics in the league or the team where was performed it passed season, of this information is not able will be done analogy with players of the league that have in the fantasy, equal price that this player, since the price in the fantasy usually is an estimate of performance expected of a respective player.

Table 1. MOW of the 15 players that integrate the squad.

	Criteria				Multiplicative factors (0 to 1)		
Player	PALS (1-40)	PA3G (1-100)	LPAT (1-60)	RPWM (1-10)	PPTW	EPTW	Total Ak
J01 (GK)	22	48	50	9	1	1	129 (02)
J02 (GK)	10	25	45	10	1	1	90 (08)
J03 (Df)	20	42	40	10	1	1	112 (04)
J04 (Df)	14	30	35	10	1	1	89 (09)
J05 (Df)	12	25	38	10	1	1	85 (11)
J06 (Df)	11	21	45	10	1	1	87 (10)
J07 (Df)	6	14	48	10	1	1	78 (14)
J08 (Mi)	22	40	50	10	1	1	122 (03)
J09 (Mi)	15	28	46	10	1	1	99 (06)
J10 (Mi)	13	25	44	10	1	1	92 (07)
J11 (Mi)	11	23	38	10	1	1	82 (12)
J12 (Mi)	6	12	52	10	1	1	80 (13)
J13 (Fo)	40	100	60	10	1	1	210 (01)
J14 (Fo)	16	30	60	7	0,9	1	102 (05)
J15 (Fo)	10	17	35	10	1	1	72 (15)

The table 1, corresponds to a MOW of the second type, with different criteria ranges, but without using multiplicative values.

In this table 1, the players have identified J01 up to J15, beginning for the GK, followed by the Df, the Mi and finally the Fo. In the column Total Ak, where the points reached by each player are reflected, to facilitate the selection, it has been included among parenthesis the order occupied by each one.

For the total points can be observed that the J13, that is a Fo, it is that of better score and enough over to the remainder of the squad. In general the Fo and the GK are used to having greater points in the fantasy that the Mi and the Df, for this case J13 is a Fo that has been playing in an excellent way in these last times.

According to the points obtained, should be placed like assets for this week: J13 (Fo), J01 (GK), J08 (Mi), J03 (Df), J14 (Fo). With these five first players already the formations 4-5-1 and 5-4-1, presented in the subsection 2.1, would remain eliminated, equally the player J02 cannot be used, since only it will be used a GK. And to complete the 11 players should be included: J09 (Mi), J10 (Mi), J04 (Df), J06 (Df), J05 (Df) and J11 (Df).

From there, that for the week illustrated in the table 1 it has a scheduling that gives a formation: 1 GK, 5 Df, 3 Mi and 2 Fo.

This assessment is repeated every week to have the weekly scheduling, obtaining together to it the formation to use that week.

In general the criterion PALS usually changes little from week to week, but if there are very notorious variations in the criteria LPAT and even in PA3G, since many players tend to have very high performance in streak of luck.

They also have very strong variations the multiplicative factors PPTW and EPTW, since sometimes the relations of the player with his coach or their team they change at any moment and in especial, for injuries that the players can suffer, further key players are usually reserved for other competitions especially when the player usually does not have a high performance against the team to face, or when it is considered that the team to face is not a strong team and can win the game without having the best players on the court.

Table 2. Percentage ranges applicable to all criteria.

Qualification	Percentage value of the weight of the criterion
Excellent	92 to 100 %
Superior	81 to 91 %
Very good	69 to 80 %
More than good	57 to 68 %
Good	45 to 56 %
Less than good	33 to 45 %
Regular	22 to 32 %
Less than regular	11 to 21 %
Bad	0 to 10 %

Importantly, although here have not used the names of the players, is a matter of real players.

From there it is interesting to observe the players J01 and J14, which although they reached the second and fifth greater quantity of points in the table 1, were the unique that did not have the 10 points of RPWM, what reflected some problems with the coach or the team.

In the case of the J14 finished being in the bench the majority of the games and surely will abandon the team, while the player J01, also was in the bench, what surely it contribute that then he was suffering an injury that will have several weeks out of game, by this reason is convenient to redo the squad and recruit new players, what can change completely the scheduling that can be obtained from that moment.

As is was already expose, there will not be a scheduling of every week of the fantasy, since the objective was to present how the MOW could collaborate to obtain the scheduling weekly and with the presented till now, it is already possible to see clearly what would be the way of working with them. With this information, already the conclusions can be presented.

5 Conclusions and futures research

As can be seen in the example realized through the table 1, the Matrixes Of Weighing (MOW), they are one of the technical multicriteria simpler to utilize and to implement. Also across this table 1, it was possible to see that there was fulfilled the objective to obtain the scheduling week per week for a fantasy soccer team.

In particular, to be able to use the multiplicative factors in MOW, it gives them a big flexibility and it permits to rule out alternatives that in a moment given do not comply with the conditions that the problem requires.

In the case here illustrated, the players J02 and J14, they will be ruled out of the scheduling weekly, when the multiplicative factors approach zero or they be finished to do zero, this aspect that would not be achieved in rapid form if only there was doing pursuit of the remaining criteria.

As future research is recommended to make a deeper use of these MOW, not only in the fantasy soccer, but in any another type of fantasy and even, being more ambitious is interesting to think about its use in the pitches. But not only the MOW are recommended for to be used in the sports in general, but also other technical multicriteria as the Multiattribute Models with multiplicative factors (MMwMf) can also be brought to the fields.

References

1. Burke, E. J. "Quasi-property" Rights: Fantasy or Reality? An Examination of C.B.C. Distribution & Marketing Inc. v. Major League Baseball Advanced Media, L.P. and Fantasy Sports Providers' Use of Professional Athlete Statistics, *Washington University Journal of Law & Policy*, 161-190 (2008).
2. Correll, J. *Player performance index for major league soccer*, Thesis for Master of Arts Economics, Clemson University, S C, USA (2012).
3. Environmental Management Systems. *Project Evaluation Matrix*, 1-44 (1997).
4. ESPN. Official Rules. Desafío La liga. In: http://games.espn.go.com/desafio-la-liga/es/story?pageName=desafio-la-liga\reglas, consulted February 2013 (2012).
5. Flockhart, D. *Fantasy soccer and Mathematics: A resource guide for teachers and parents. The innovative Math program based on real-life Sports Statistics*, Jossey-Bass book, CA, USA (2007).
6. García G., M. J. and Hernández R., J. G. Multiattribute Model with Multiplicative Factors and Matrixes Of Weighing and the Problem of the Potable Water, In R. Espin, J. Gómez & A. Racet (Ed), *Towards a Trans-disciplinary Technology for Business Intelligence Gathering Knowledge Discovery, Knowledge Management and Decision Making*. Shaker Verlag: Germany, 364-375. (2011).
7. García G., M. J., Hernández R., J. G., and Hernández G., G. J. Una Metodología Integradora-Adaptable para desarrollar Sistemas de Apoyo a las Decisiones (MIASAD), In J. Valderrama (Ed.), *Congreso Interamericano de Computación Aplicada a la Industria de Procesos CAIP'2011*. Universidad de Girona, España, 753-760 (2011).
8. Hamscher, V., Schwiegelshohn, U., Streit, A. and Yahyapour, R. Evaluation of Job-Scheduling Strategies for Grid Computing, In 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000), Berlin, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Heidelberg, Germany, 191-202 (2000).
9. Hernández R., J. G., García G., M. J. and Hernández G., G. J. Matrixes Of Weighing and catastrophes, *International Journal of Distributed Systems and Technologies*, 2(1), 14-28 (2011).
10. Hernández R., J. G., García G., M. J. and Hernández G., G. J. Dynamic knowledge: Diagnosis and Customer Service, In N. Delener (Ed.), *Service Science Research, Strategy, and Innovation: Dynamic Knowledge Management Methods*. IGI Global: Hershey, PA USA, 547-573. (2012).
11. Massari, M. G. When fantasy meets reality: The clash between on-line fantasy sports providers and intellectual property rights, *Harvard Journal of Law & Technology*, 19(2), 443-465 (2006).
12. Oates, T. P. New Media and the Repackaging of NFL Fandom, *Sociology of Sport Journal*, 26, 31-49 (2009).
13. Pillay, U., Tomlinson, R. and Bass, O. *Development and dreams. The urban legacy of the 2010 football world cup*, HSRC Press, South Africa (2009).
14. Prasad B. JIT quality matrices for strategic planning and implementation, *International Journal of Operations & Production Management*, 15(9), 116-142 (1995).
15. Shen, S. and O'Hare, G. M. P. Agent-Based resource selection for grid computing, *Workshop 4: International Workshop on Agents and Autonomic Computing and Grid Enabled Virtual Organizations*, 658-672 (2004).
16. Taylor, P., Funk, C. and Craighill, P. Americans to rest of world: Soccer not really our thing, *Pew Research Social & Demographic Trends*, 1-10 (2006).

MISTA 2013

A computational study of a stock cutting problem with sequence dependent cut losses

Michele Garraffa · Fabio Salassa · Wim Vancroonenburg · Greet Vanden Berghe · Tony Wauters

1 Introduction

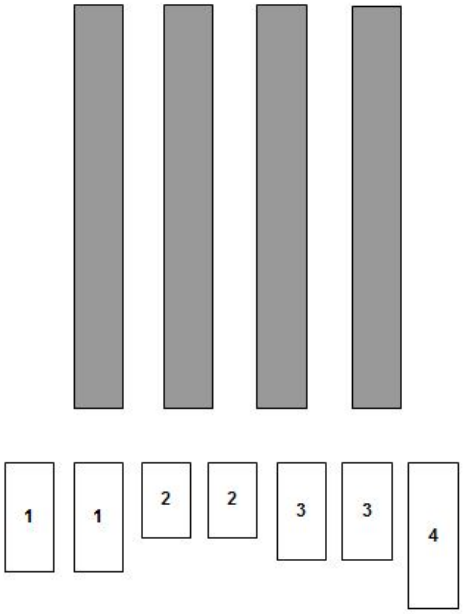
Cutting stock and bin packing problems arise in many different fields such as in textile, glass, steel, wood and paper industries. The main objectives are usually to minimize the material waste and/or to minimize the amount of raw material needed for satisfying the demand. Many papers on the 1-dimensional cutting stock problem (1D-CSP) refer to material waste as leftovers in the cutting patterns. But these are not the only material losses which occur in practical contexts. In joineries, for instance, another type of material loss is caused by the cuts of the blade. When the overall waste is negligible, these cut losses are not taken into account. They are not always small however, especially when the material is cut on different angles. For example, door and window profiles are typically cut under either an angle orthogonal to the longitudinal axis of the profile, or under an angle of 45 degrees. Since cutting a material under different angles incurs different losses, the sequence in which items are cut determines the total cut loss. The present contribution aims at describing all the features of a 1-dimensional cutting stock problem with such sequence dependent cut losses (1D-CSP with SDCL), introducing efficient algorithms and analysing their performance with respect to particular instance characteristics.

2 Problem description

An instance of the problem is formed by a set I of n input items, each one with a demand d_i and a size s_i, and by a matrix $\boldsymbol{C}$ $((n+1)\times(n+1))$ whose components $c_{i,j}$ indicate the cut loss if the item i is cut before j. The parameters $c_{0,i}$ and $c_{i,0}$

Michele Garraffa · Fabio Salassa
DAUIN, Politecnico di Torino
E-mail: michele.garraffa@gmail.com,fabio.salassa@polito.it

Wim Vancroonenburg[ab] · Greet Vanden Berghe[ab] · Tony Wauters[a]
[a]CODeS, KAHO Sint-Lieven
[b]CODeS, KU Leuven KULAK
E-mail: {wim.vancroonenburg, greet.vandenberghe, tony.wauters}@kahosl.be



$$\begin{bmatrix} 0 & 2.4 & 2.5 & 1.9 & 2.6 \\ 1 & 2.5 & 2.5 & 1.5 & 2.7 \\ 1.3 & 1 & 3 & 2.8 & 3 \\ 3.5 & 0.5 & 0.2 & 2.5 & 2.7 \\ 2.2 & 2.8 & 1.4 & 3 & 2.5 \end{bmatrix}$$

(a) Cut losses matrix, with c_{ij} the cut loss when item j is cut after item i

$\boldsymbol{i}$	**1**	**2**	**3**	**4**
$\boldsymbol{l_i}$	4.5	3.1	4	6
$\boldsymbol{d_i}$	2	2	2	1

(b) Length and demand for each item type

Fig. 1: Example of an input instance

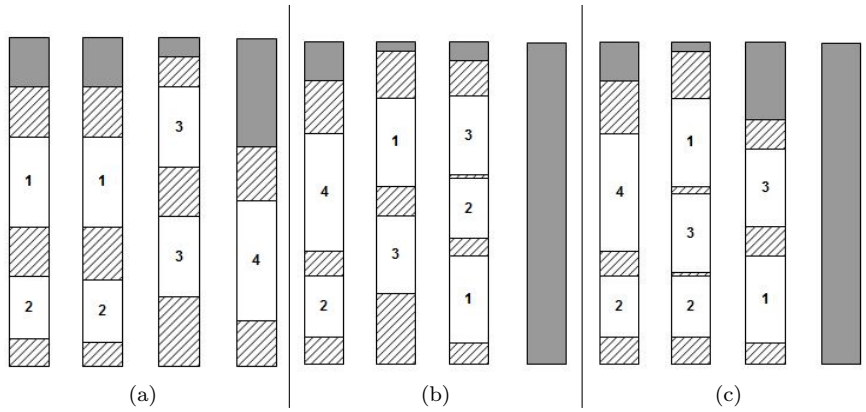


Fig. 2: Three solutions of the instance in Fig.1. First solution (2a), 4 raw materials used. Second solution (2b), 3 raw materials used. Third solution (2c), 3 raw materials used and smaller losses.

indicate the cut losses which occur respectively when i is the first or the last item in the raw material to be cut. The problem is to find a minimal set of feasible patterns, to be cut from raw material of length L, that cover the overall items' demand. As a secondary objective, the aim is to find the set of cutting patterns that is most reusable in terms of leftovers.

According to Dyckhoff's typology [1], this problem can be classified as 1/V/I/M or 1/V/I/R depending on the assortment of items. In fact, it is a 1-dimensional problem with an unlimited supply of objects (raw materials) of identical size and a set of orders (items to cut). Using the typology introduced by Wäscher et al. in [5], the problem can be classified as a Single Stock Size Cutting Stock Problem (SSSCSP). The literature on this particular class of cutting stock problems is vast, starting as early as the 1960's with the work of Gilmore and Gomory [3]. However, the added feature of sequence dependent cut losses makes the problem more complicated. Restricting the feasible cutting patterns to subsets of items that have a feasible sequence of at most length L, implies solving some form of a TSP problem.

Fig.1 reports an example instance for which input data are provided by Tables 1a and 1b. Fig.2a, Fig.2b and Fig.2c provide three solutions of this instance. The cut losses are drawn with a dashed background, while the leftovers are shown in grey. These figures illustrate that these losses are in the same order of magnitude of the item sizes and have got a very high variability. The example shows that changing the order in which the items are cut determines a different amount of material waste due to the cut losses. The raw materials used in Fig.2b are fewer than in Fig.2a as a result of the smaller amount of material waste. However, the same instance of the problem may admit a high number of solutions which consume the same amount of raw materials. Fig.2c shows another solution with 3 raw materials. Careful analysis of the solutions in Fig.2b and Fig.2c indicates that the one exposed in Fig.2c is the most suitable for industrial purposes. In fact, the average size of the leftovers is larger than the one in the solution illustrated in Fig.2b. This implies that a higher amount of material is reusable for successive orders. Definitely, large leftovers are easier to reuse than small or medium leftovers. For this reason, we consider as a second objective, once the amount of raw materials is minimized, to maximize the sum of the squared leftovers produced by the solution. The square function makes it possible to obtain leftovers which differ considerably in size in order to provide a high number of large reusable leftovers. This objective was adopted in analogy with [2], where maximizing the sum of square loads on a bin packing problem allows to achieve the optimal space utilization.

2.1 Connections with other problems

1D-CSP with SDCL presents some interesting analogies with other combinatorial optimization problems. The trivial case in which all the cut losses are negligible, naturally leads to the common one dimensional bin packing problem (1D-BPP). Moreover, the same analogy persists whenever the variability of the cut losses is insignificant. Let $\bar{c} = c(i,j)\ \forall i,j \in I_0\ (= I \cup \{0\})$ be the value of every cut loss. In this case the problem can be seen as a 1D-BPP where the bin size is equal to $L - \bar{c}$ and the size of each item is increased by $\bar{c}$.

The problem can also be modelled as a distance constrained vehicle routing problem (DCVRP or DVRP). The three index formulation in [4], named VRP4, can be adapted to model 1D-CSP with SDCL without considering the capacity constraints of the vehicles and by minimizing the number of vehicles. If the second objective function previously discussed has to be adopted however, the overall model becomes nonlinear. In this context, the raw materials and the items of the

1D-CSP with SDCL are represented respectively by the vehicles and the cities of the DVRP, while the dummy item 0 corresponds to the depot in vehicle routing. The cut losses matrix C whose components are $c(i,j)$ $\forall i,j \in I_0$ is replaced by an analogous matrix C' that indicates the distances between cities, increased by (half of) the respective item lengths. However, it must be noted that no assumptions can be made on the symmetry or planarity of the distance-matrix C'. In fact, the only plausible assumption about the cut losses is that their values are positive. This makes C more general and the VRP heuristics which take advantage of these properties are therefore not directly applicable to 1D-CSP with SDCL.

3 Computational study

We will present a computational study that investigates the relevance of considering such sequence dependent cut losses. To this end, a tailored pattern based heuristic is presented. Each step of the proposed algorithm generates an opportune number of convenient patterns, then the current demand is covered to optimize the objectives, thanks to an exact set covering formulation. The patterns are generated in order to guarantee the feasibility of the solutions computed at each step. A part of this solution is fixed and another step is performed on the remaining demand, until there are no other items to cut.

The performances are compared with both classic bin packing and VRP heuristics, as well as to a state-of-the-art bin packing heuristic on a set of instances with varying characteristics. Some real world instances, kindly provided by TECHWIN Software, are included with the aim to prove the practical importance of the problem. The main characteristics considered are the ratio of average item length w.r.t. the raw material length L, the ratio of the average cut loss w.r.t. the average item size and finally the variability of the cut losses. Initial experimentation confirms that the dedicated algorithm has a clear advantage whenever the instances are less similar to bin packing instances (i.e. when losses are large and have high variability), but also when the item length is small w.r.t. the raw material length. The latter indicates that the applicability of the problem formulation also extends to those bin packing instances where losses were assumed negligible.

References

1. H. Dyckhoff. A typology of cutting and packing problems. European Journal of Operational Research. 44: 145-159, 1990
2. K. Fleszar, K. S. Hindi. New heuristics for one-dimensional bin-packing. Computers & Operations Research. 29: 821-839, 2002.
3. P.C. Gilmore, R.E. Gomory. A linear programming approach to the cutting-stock problem-Part II. Operations Research 11, 864888, 1963.
4. G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research. 59: 345-358, 1992.
5. G. Wäscher, H. Hauner, H. Schumann. An improved typology of cutting and packing problems. European Journal of Operational Research. 183: 1109-1130, 2007.

MISTA 2013

Scheduling with MS Project: an add-in

André Filipe Macedo Guedes • Anabela Pereira Tereso

1 Introduction

The main objective of this research was to integrate scheduling techniques, documented in the literature, with Microsoft Project 2010, in order to help project managers' deal with scheduling tasks in an easier and controlled way, and at the same time achieve better results.

As far as we know, there is no information about the techniques or algorithms used by Microsoft Project to do the scheduling tasks so the possibility to select the algorithms to use gives a higher degree of control to the project manager.

The initial hypothesis was that it is possible to get better results, concerning the project's duration, using the studied techniques rather than the default scheduling technique used by Microsoft Project 2010, with an increased control.

2 Project Scheduling and Scheduling Techniques

The most supported motto, by a vast majority of authors, is that project scheduling is the most important task for the success of a project. The more detailed and clear the scheduling is, the easier will be the progress of the project. There will be greater interconnection between the players involved, with consequent maximization of the efficiency of resources utilization, minimization of costs, and higher profits.

But what is, exactly, project scheduling? Scheduling is to set the guidelines that the project team has to follow to achieve success. These guidelines indicate which tasks need to be done and which resources are needed to implement them. They help to meet the customer's expectations concerning delivery dates, cost and work accomplished. The ultimate goal of a project manager is to fulfill these expectations in an optimal way, if possible.

Project scheduling, despite being detailed and objective, can be reduced to three questions: what, how and when do things need to be done?

The most important aspect, regarding the scheduling phase, is the actual tasks' schedule. The scheduling is a way to indicate the sequence in which the activities of a project will be implemented. From this sequence of events, the project manager will be able to make estimations regarding the time of beginning and ending of each activity. These values will be crucial, so that the goal for project completion, established with the client, can be met.

But building this sequence of activities is not a trivial task. To be able to proceed with the scheduling of the project's activities it is necessary to take into account a range of details and constraints that will directly impact on this task. Each activity is characterized by an amount of time needed for its execution (the duration), an amount of resources needed to support the implementation and a set of precedence relations with other activities. All these parameters are important when allocating an activity. It can't be scheduled to run when there are insufficient resources or when one of their predecessor activities, if any, is not yet completed. This problem is known in the literature as the Resource Constrained Project Scheduling Problem (*RCPSP*).

A simple solution for the scheduling problem, without considering resource constraints, would be to make use of the slack of an activity to move its execution time to a moment that does not interfere with another conflicting activity, nor the total project time, whenever possible.

André Filipe Macedo Guedes
Anabela Pereira Tereso
CGIT Research Center
University of Minho, 4800-058 Guimarães, Portugal
afm.guedes@gmail.com; anabelat@dps.uminho.pt

The *RCPSP* aims to find the time instants in which the activities of a given project should begin, subject to precedence and resource constraints, in order to minimize the project duration.

Due to the precedence relationships, it is common for an activity a_j to have more than one predecessor. If these predecessors do not finish at the same time, activity a_j will have to wait until the last one finishes, before it can start its execution. This makes the activities that have ended earlier as predecessors to have slack time.

The scheduling techniques used in the developed add-in were: Early Start Schedule, Late Start Schedule, Constructive Heuristics (with different Priority Rules, Scheduling Schemes and Scheduling Directions) and Branch-and-bound (see references [1-6] for more details on these techniques).

3 MS Project Add-in

The MS Project Add-in was developed using the Microsoft Visual C Sharp (C#) programming language in the Integrated Development Environment Microsoft Visual Studio 2010 (VS2010) on the Windows 7 operating system.

Figure 1 shows the tab of the software application developed that appears within MS Project, with four buttons that allow the end user to choose the algorithm and perform the scheduling.

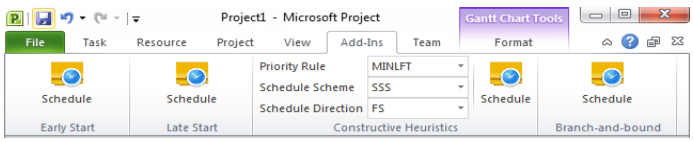


Figure 1 – Tab with the new MS Project Add-in

After the user presses the OK button, the algorithm will compute the start and end times of all activities, and will adjust the Gantt chart to represent these changes.

In addition to the treatment of the data from MS Project files, entered directly by the user, it is also possible to upload projects in the test environment from VS2010.

4 Results

To allow performing tests to the developed add-in, some project examples were created. The different methods studied were tested with all these projects. Due to the extension of the results obtained and to limitations of space, only one illustrative project will be presented in this paper.

The results presented are the Gantt charts obtained and the total project duration, for each scheduling technique implemented.

The Early Start Schedule was tested using some example projects and the schedules obtained were identical the ones generated by MS Project without resource constraints, so they are not presented in this paper.

The Late Start Schedule also gave the same results as MS Project, if the user selects to schedule activities as late as possible.

Figure 2 represents the example project network with ten activities and three resource types, which will serve to illustrate the application of the rest of the scheduling techniques. The numbers above the activities represent the activity duration and the numbers below the activities represent the number of resources needed.

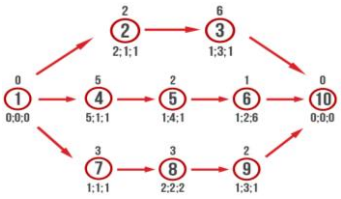


Figure 2 – AoN network for example project

The following resources availabilities were considered:

$$a_1 = 6, a_2 = 4 , a_3 = 8.$$

For the thirty two possible combinations to apply the *Constructive Heuristics*, a great variety of results were observed. Some shown a reduction on the total duration of the project, some gave similar results to those of MS Project (using manual resource leveling), and some shown worst results.

Figure 3 shows a typical result obtained using one of these scheduling techniques. Figure 4 shows the result obtained by MS Project, so a more objective comparison can be made.

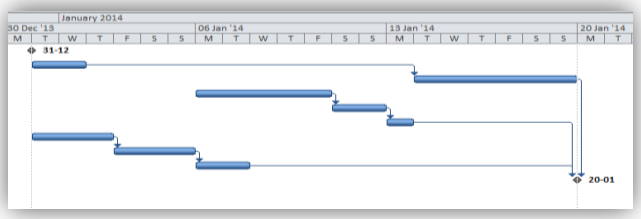


Figure 3 – Scheduling obtained using *Constructive Heuristics* (SPT – SSS – FS)

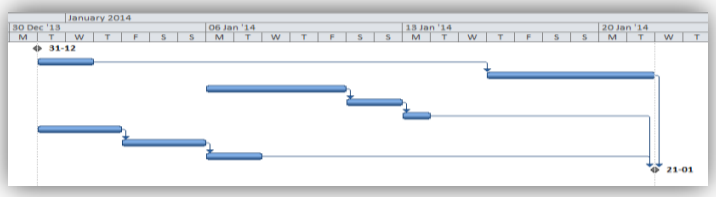


Figure 4 – Scheduling obtained using MS-Project with manual resource leveling

The result of the application of the *Branch-and-Bound* technique is presented in Figure 5. It can be seen that it represents a good alternative to the other schedules.

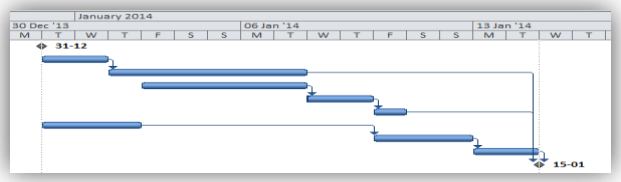


Figure 5 – Scheduling obtained using *Branch-and-Bound*

5 Conclusions

After performing all the tests, with various project examples, the conclusion was that *Branch-and-Bound* is the technique that achieves better results. For the example project,

Branch-and-Bound obtained a schedule with duration equal to 15 days, while MS Project reached the minimum of 21 days.

In projects with one or two different types of resources, and few critical paths, the results do not show a great variation between methods. This is due to the low complexity of the project networks, which results in schedules that do not differ much. Results show significant differences for projects that have a large number of activities. For these projects, the schedule computed by MS Project is dominated by almost all the methods used.

The best results belong to the *Branch-and-Bound* method and *Constructive Heuristics*, composed by *Parallel Scheduling Scheme*, the priority rule *MINLFT* and the *Backward Direction*. The worst outcome is for the heuristic using the *Serial Scheduling Scheme*, the priority rule *GRD* and the *Forward Direction*.

As the *Branch-and-Bound* method is an implicit enumeration technique, all possible solutions to the problem were analyzed, and it was chosen the one having a shorter duration, since this is the variable to be minimized.

Thus, the conclusion was that the resource leveling method used by MS Project to schedule a project under resource constraints can be used for simple situations, where there are a small number of activities and few different types of resources. Even in these cases, its performance is poor and it is not automatic.

When it comes to larger and more complex networks, with regard to the variety of resources available, the technique used by MS Project is overtaken by the majority of the methods studied, mainly by *Branch-and-Bound* and the *Constructive Heuristics* using the *PSS*, *MINLFT* and *Backward Direction*. Another important outcome is the possibility made available to the user to select the scheduling method he wishes to use, allowing more control on the results obtained.

There are a number of other approaches to the RCPSP that should be interesting to address in the future, namely the use of meta-heuristics, like the pseudo particle swarm optimization, recently proposed to solve this problem [7].

Acknowledgements This work was supported by Portuguese Foundation for Science and Technology, under Project Pest-OE/EME/UI0252/2011.

References

1. Demeulemeester E, Herroelen W. Project Schedule - A Research Handbook: Kluwer Academic Publishers; 2002
2. Brucker P, Drexl A, Mohring R et al. Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 1999; 112: 3-41
3. Ulusoy G, Ozdamar L. Heuristic Performance and Network Resource Characteristics in Resource-Constrained Project Scheluling. Journal of the Operational Research Society 1989; 40: 1145-1152
4. Davis EW, Patterson JH. A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling. Management Science 1975; 21: 944-955
5. Boctor FF. Heuristics for scheduling projects with resource restrictions and several resource-duration modes. International Journal of Production Research 1993; 31: 2547-2558
6. Brucker P, Drexl A, Möhring R et al. Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 1999; 112: 3-41
7. Nasiri MM. A pseudo particle swarm optimization for the RCPSP. International Journal of Advanced Manufacturing Technology 2013; 65: 909-918

MISTA 2013

Modeling Real-World Vehicle Routing Problems with Dependencies

Thijs Walcarius · Maarten Houbraken · Pieter Audenaert · Mario Pickavet · Piet Demeester

1 Introduction

In this research, we consider a novel data model that can be used to solve the Vehicle Routing Problem (VRP) with several of its specializations encountered in the real world.

The basic *Vehicle Routing Problem* is an optimization problem in which a number of customers have to be served with a fleet of vehicles, while minimizing the cost for doing so. As this problem has many applications in the fields of transportation, logistics and distribution, several specializations exist. The *Vehicle Routing Problem with Dependencies* (VRPD) adds restrictions to the order in which customers are served. A dependency between customers indicates a "must happen before"-relationship: $t_{c1} < t_{c2}$. The *Capacitated Vehicle Routing Problem* (CVRP) constrains the maximum amount of cargo that a vehicle can carry at any time. The *Vehicle Routing Problem with Pick-up and Delivery* (VRPPD) is a further specialization of the CVRP and the VRPD, in which cargo is picked up at a customer's location, and has to be delivered to another customer. The delivery event can only happen after the pickup event has been completed. The *Vehicle Routing Problem with Time Windows* (VRPTW) is a variation in which each event has a time window in which it has to be completed: when the vehicle arrives before the earliest allowed time, it has to wait at that location until the time window opens. Ending an event after the time window is prohibited. In the basic VRP, all vehicles start and end at the same location, called the depot. In the *Multiple Depot Verhicle Routing Problem* (MDVRP), this restriction is loosened, and each vehicle can have its own location.

Most current research focuses on modeling the VRP and its extensions as an *Integer Programming* (IP) problem. In this research, we propose a novel approach, in which the problem is modeled in a directed acyclic graph (DAG). Combined with heuristics that incorporate domain knowledge of the VRP and its specializations, it proves to be an efficient approach.

Thijs Walcarius, Maarten Houbraken, Mario Pickavet and Piet Demeester
IBCN, INTEC, Ghent University, Belgium - iMinds
E-mail: `firstname.lastname`@intec.ugent.be

2 Modeling the VRP as a DAG

Let $G = (V, A)$ be a directed graph. Each vertex v represents an event, consisting of a location $Loc(v)$ and duration $D(v)$. It can also contain a timestamp t_v, that signifies when the event is executed. When the event isn't part of the planning, this timestamp field is left empty. Depending on the type of VRP-problem being solved, additional properties can be added. In our model, there are two types of vertices: *event vertices* and *depot vertices*. An *event vertex* v represents a customer being serviced. In the CVRP it has a capacity delta Δ_v. A *depot vertex* represents the depot where the vehicle leaves from and returns to. As we try to model the problem as an acyclic graph, each depot is duplicated into two *depot vertices*: each tour starts in a *depot start vertex* and ends in a *depot end vertex*.

An edge $e = \langle v_1, v_2 \rangle$ between two vertices denotes a relationship between two events. In our DAG model, we define several types of edges to denote the various relationships between events. A planning edge depicts the most important connection between two events: when a vehicle drives from event e_1 to event e_2, a planning edge is added from v_{e1} to v_{e2}. As this edge signifies a movement of a vehicle, it has a duration $D(e)$: the time it takes to get from $Loc(v_{e1})$ to $Loc(v_{e2})$. In the CVRP, this edge also has a load l_e. This load must be between 0 and the maximum capacity of the vehicle. For the VRPD, a second edge type is introduced: the *dependency edge*. This edge signifies that the event e_1 must happen before the event e_2: $t_{v1} + D(v_1) < t_{v2}$.

For the VRPTW, we enhance the DAG by adding two time windows to each vertex: the *given time window* and the *actual time window*. The *given time window* $[\check{t_v}, \hat{t_v}]$ indicates the earliest start time and the latest end time for the event. For an *event vertex*, this is defined by the customer. In a *depot vertex*, this time window is equal to the operation hours of the vehicle. The *actual time window* $[\check{\tau_v}, \hat{\tau_v}]$ is initially equal to the given time window, but is further constrained by the incoming and outgoing edges. The minimum start time imposed by an edge $e = \langle v_1, v_2 \rangle$ can be computed as follows: $MST(v_2) = MST(v_1) + D(v_1) + D(e)$. Computing the maximum end time can be done analogously. When one of the bounds of the time window changes, this has to be propagated throughout the DAG. Because no cycles are allowed, this updating process cannot result into infinite loops. The window can never become smaller than the duration of the event, ensuring that the planning remains feasible.

2.1 Usage of the DAG model

At first, the DAG has to be initialized. For every customer that has to be served, an *event vertex* is created with an empty timestamp. For each vehicle that is available, two *depot vertices* are added. Between these two vertices, a planning edge is added with duration 0.

To add an event c to the route of a vehicle between event a and b, the planning edge $\langle v_a, v_b \rangle$ is removed, and two new planning edges $\langle v_a, v_c \rangle$ and $\langle v_c, v_b \rangle$ are added. The durations of these edges are efficiently calculated via a separate routing engine. Additionally, the timestamp of vertex v_c is set, or in the case of the VRPTW, the bounds of the time windows are updated: the maximum end time is recomputed for vertex a and c, the minimum start time is recomputed for vertex b and c.

2.2 Abstracting orders and tasks

In real-world scenarios, logistic companies have to deal with orders that consist of multiple events. Either an order is added to the planning, and all associated events have to be executed, or it is held back and none of the events are executed. When two or more events have to be executed by the same courier (ex. a pickup and delivery of a package), we call them a *task*. One order can consist out of multiple tasks, for example when a package is stored in an intermediate storage and delivered by another courier.

The proposed model only contains events, and makes abstraction of tasks and orders. These are however inherently present in the DAG, as events of the same task typically have a dependency between them. When an order contains multiple tasks, the event of these tasks will also be connected by a dependency edge.

The insertion and removal heuristics used are responsible for inserting and removing all related events to a planning. When events of an order are partially added to a planning, their timing will immediately be reflected in the *actual time windows* of all events that are connected through dependencies. When a partial planning of an order makes the planning infeasible to be executed, this will become clear as the *actual time window* cannot be made smaller than the duration of that event.

3 Planning Heuristics using the DAG

Since the formulation of the VRP[1], a tremendous amount of research has been done[2]. We are currently in the process of trading off the most promising heuristics that can be adapted for usage with the DAG described above. Adopting insertion heuristics is currently the most promising approach, as they can benefit from the efficient updating of the *actual time windows*, instead of having to perform extensive checks. Next to that, optimization heuristics like Lin-Kernighan[3] can also benefit from the proposed model, but require multiple operations on the DAG. By employing a styled version of the DAG structure, we focus on efficient implementations of the most occurring primitive operations, trading off execution time with less occurring primitives.

In our approach, we pay attention mostly to the implementation details of the different algorithms and data structures. Careful implementation and corresponding design techniques do pay off dividends, even in the short term. For the current project, further research is needed to determine how this affects the performance of the techniques described above.

Acknowledgements This research was conducted with support from IWT, iMinds, Ghent University and a private company.

References

1. Dantzig, G.B., Ramser, J.H.: The truck dispatching problem. Management science **6**(1), 80–91 (1959)
2. Gendreau, M., Potvin, J.Y., Bräumlaysy, O., Hasle, G., Løkketangen, A.: Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. The vehicle routing problem: Latest advances and new challenges pp. 143–169 (2008)
3. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Operations research **21**(2), 498–516 (1973)

MISTA 2013

New Solvable Cases of the Maximum Lateness Problem with Preemptions

Sam Walker · Yakov Zinder

1 Introduction

This paper introduces a new class of partial orders (or equivalently, acyclic directed graphs) and shows that this class in certain sense determines when the algorithm, presented in [8], can construct an optimal schedule for the maximum lateness problem with parallel identical machines and partially ordered tasks, allowing preemptions. The considered class broadens the confines of the realm of known solvable cases, i.e. cases when an optimal schedule for the maximum lateness problem can be constructed in polynomial time.

The considered algorithm is one of the three algorithms for the maximum lateness problem with parallel identical machines and partially ordered tasks, allowing preemptions, [3], [8] which are known to the authors. Each of these three algorithms has a counterpart intended for the maximum lateness problem with parallel identical machines and partially ordered unit execution time tasks [1], [2], [7]. The choice of the algorithm for the analysis in this paper was dictated by a superior performance of the algorithm in [7] in comparison with the algorithms in [1] and [2] (see for example [6] for more information).

The maximum lateness problem with parallel identical machines and partially ordered tasks, allowing preemptions, can be stated as follows. A finite set of tasks $N = \{1, 2, \ldots, n\}$ is to be processed on $m > 1$ identical parallel machines. The processing of tasks begins at time $t = 0$. Each machine can process only one task at a time. The order in which tasks can be processed is restricted by precedence constraints – a transitive, antireflexive and antisymmetric relation on N. If task j precedes task g, denoted $j \to g$, then task g cannot be processed until task j has been completed. In order to be completed, a task j requires p_j units of processing time. At any point in time, the processing of a task can be interrupted and resumed later on the same or a

Sam Walker
University of Technology, Sydney
E-mail: samuel.j.walker@uts.edu.au

Yakov Zinder
University of Technology, Sydney
E-mail: yakov.zinder@uts.edu.au

different machine. The goal is to find a schedule minimising the maximum lateness

$$L_{max}(\sigma) = \max_{j \in N}\{C_j(\sigma) - d_j\},$$

where $C_j(\sigma)$ is the completion time of task j in schedule σ, and d_j is the point in time (normally referred to as due date or due time) by which it is desired to complete task j. In the three field notation (see for example [4]), the considered problem is denoted by $P|prmp, prec|L_{max}$.

2 The algorithm and the definition of its domain

It is known that the problem stated above is NP-hard [5] and that the three known algorithms for the $P|prmp, prec|L_{max}$ problem are capable of constructing an optimal schedule only in some particular cases. Not much is known about these solvable cases. The only available information is that all three algorithms construct an optimal schedule if the partially ordered set of tasks is an in-tree, i.e. each task has at most one successor, and two of them also construct an optimal schedule for arbitrary precedence constraints when $m = 2$. The next section provides a description of the set of all partial orders which has the following property:

- for any partial order in this set and any task durations and due dates, the algorithm presented in [8] constructs an optimal schedule;
- for any partial order which is not in this set, there exist such task durations and due dates that the algorithm presented in [8] fails to construct an optimal schedule.

The above mentioned set of partial orders will be referred to as the domain of the considered algorithm.

The considered algorithm constructs the schedule in several stages, each corresponding to a so-called allocation point (a point in time where a decision on machine allocation is made). The machine time is allocated to the available tasks according to the tasks' priorities which are calculated as $p_j(t, \sigma)+\mu_j$, where $p_j(t, \sigma)$ is the remaining processing time of task j at time point t in schedule σ, and μ_j is a constant computed as a maximum between $-d_j$ and an upper bound on the maximum lateness for the maximum lateness problem induced by the successors of j, i.e. the problem where all tasks are successors of j with the same precedence constraints and due dates. This upper bound is obtained by applying the same algorithm to the induced problem.

If the number of tasks at an allocation point is greater than m, then some tasks may be processed between this and the next allocation points for the period less than the duration of the time interval between these allocation points, whereas some other tasks may not be processed between these two consecutive allocation points at all. The allocation of machine time is conducted in such manner that the resultant schedule has the following three properties: (1) if two tasks are available at an allocation point and the priority of one task is greater than or equal to the priority of another, then this inequality between priorities holds at the next allocation point; (2) a machine can be idle between two consecutive allocation points only if the number of all available tasks is less that m and each such task is continuously processed on the time interval between these allocation points; and (3) a task can be allocated processing time at an allocation point only if each available task of higher priority is allocated at this point the entire interval between this and the next points of allocation.

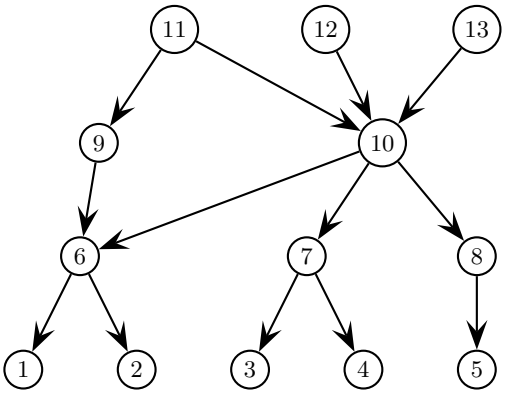
3 The domain

In what follows, the terms task and node, as well as the terms partial order and acyclic directed graph are equivalent. Correspondingly N will denote the set of tasks as well as the set of nodes. A number of classes of acyclic directed graphs (classes of partial orders) which are frequently used in scheduling theory, such as in-trees, out-trees, series-parallel graphs, and interval orders, can be described by specifying certain prohibited subgraphs. The new class of acyclic directed graphs, constituting the domain of the considered algorithm, also will be described by some prohibited objects. The description of this class requires several definitions.

- Node j and node g in an acyclic directed graph are independent if neither $j \to g$ nor $g \to j$. A set of nodes (tasks) is independent if all pairs of nodes in this set are independent.
- The width of an acyclic directed graph is the largest cardinality among all independent sets in this graph.
- An acyclic directed graph is splittable if it can be presented as a series composition of its two nonempty subgraphs. In other words, the set of nodes (tasks) can be presented as a union $N_1 \cup N_2$ such that $N_1 \neq \emptyset$ and $N_2 \neq \emptyset$, and $j \to g$ for any $j \in N_1$ and any $g \in N_2$.
- Let X be an arbitrary non-empty subset of N. A subgraph with the node set Y is induced by X if and only if
  - $X \subseteq Y$;
  - for any $g \in Y \setminus X$, there exists $j \in X$ such that $j \to g$; and
  - for any $g \in Y$, any $q \in Y$ and any $j \in N$, the relation $g \to j \to q$ implies $j \in Y$.
- A pair of nodes is regular if and only if these nodes are independent and any subgraph induced by this pair and having width greater than or equal to four is splittable.
- A set of three or more independent nodes is regular if and only if all pairs in this set are regular.
- A set of two or more independent nodes is singular if it is not regular.

Theorem 1 *The domain of the considered algorithm is the set of all acyclic directed graphs (partial orders) that do not contain a singular set of cardinality four.*

The graph below belongs to the domain of the considered algorithm.



This graph contains several singular pairs (for example, pair 6 and 7 or pair 11 and 12) but does not contain a singular set of cardinality four. Another example is any in-tree. The following theorem shows a simple way of constructing more examples.

Theorem 2 *The domain of the considered algorithm is closed with respect to series composition.*

References

1. P. Brucker, M. R. Garey and D. S. Johnson, "Scheduling Equal-Length Tasks Under Tree-like Precedence Constraints to Minimize Maximum Lateness", Mathematics of Operations Research, Vol. 02, pp. 275-284 (1977)
2. M. R. Garey D. S. Johnson, "Two Processor Scheduling with Start Times and Deadlines", SIAM Journal on Computing, Vol. 06, pp. 416-426 (1977)
3. E. L. Lawler, "Preemptive Scheduling of Precedence-Constrained Jobs on Parallel Machines", In M.A.H. Dempster, J.K. Lenstra, and A.H.G. Rinnooy Kan (Eds), Deterministic and Stochastic Scheduling, D. Reidel Publishing Company, pp. 101-123 (1982)
4. M. Pinedo, "Scheduling theory, algorithms, and systems". Springer, New York ; London (2008)
5. J. D. Ullman, "NP-complete scheduling problems", Journal of Computer and System Sciences, Vol. 10, pp. 384-393 (1973)
6. Y. Zinder, "The Strength of considered algorithms", In proceedings of the 3rd Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2007), 28 -31 August 2007, Paris, France, pp. 531-537 (2007)
7. Y. Zinder, D. Roper "An Iterative Algorithm for Scheduling Unit-Time Tasks with Precedence Constraints to Minimise the Maximum Lateness", Annals of Operations Research, Vol. 81, pp. 321-340 (1998)
8. Y. Zinder, G. Singh, "Preemptive Scheduling on Parallel Processors with Due Dates", Asia-Pacific Journal of Operations Research, Vol. 22, pp. 445-462 (2005)

MISTA 2013

An unrelated parallel machine scheduling problem with auxiliary resources and sequence dependent setup times

Abdoul Bitar · Stéphane Dauzère-Pérès · Claude Yugma

1 Introduction

The semiconductor manufacturing process is probably the most complex process in existence ([1]). Semiconductor wafer fabrication (fab) can be described as a multistage process with re-entrant flows. The processing is done layer by layer. Each layer requires several steps of processing such as chemical mechanical polishing, diffusion, film deposition, photolithography, implant (doping) and etching. For each of the product types, and depending on the technology, a wafer goes through more than 400 process steps over a period of a few weeks. Wafer scheduling is a complex task due to the large number of products and machines involved. It is further complicated by additional constraints such as the re-entrant flow of operations, time constraints to avoid contamination, setup issues, preventive maintenances and random machine breakdowns. The importance of scheduling to the performance of semiconductor wafer fabrication facilities (fabs) has been known for many years, see for example Gupta et al. [1].

In this paper, we focus on the photolithography workshop which is generally the bottleneck area of the fab. A lot of *wafers*, called a job, has to be processed on non-identical parallel machines. A lot has at most 25 wafers, joined together to enter a machine. The process of a lot in photolithography consists of transferring an integrated circuit pattern on a wafer. In order to realize this operation, some resin is put on the wafer which is exposed to an ultraviolet ray. At this step, the use of an auxilliary resource i.e. a specific *mask* (or reticle) is necessary to shape the pattern on the wafer. The machine processing depends on both the job and the right mask being available at the same time. Considering the fact that a mask must be on the tool for the duration of a jobs processing, the relatively small number of masks present in a wafer fab further complicate the parallel machine scheduling problem. Moreover, the machines need a specific configuration, named *batch-config* to execute a lot; then switching from one batch-config to another requires a setup time, depending on the previous lot, the next lot and the machine that will execute both lots. Finally, the machines have a *qualification* or (eligibility machines) only for a set of lots (which means they can not process the other ones) and they do not always have the same processing times for the same lot. Hence, our objective is to schedule jobs on unrelated parallel machines with auxiliary resources and sequence and machine dependent setup times with three objectives described in Section 2.

In the literature, Uzsoy (2000) investigate the photolithography work center scheduling problem The authors assume that an operation which requires a specific mask type can be assigned to at most r machines, where r is the existing number of masks of the corresponding type that are available in their simulation-based analysis. Several heuristics for stepper and mask scheduling based on appropriate modifications of the Apparent Tardiness Cost (ATC) dispatching rule are described by de Diaz et al. [2]. An integer linear program and scheduling heuristics based on dispatching rules for the problem $P_m|r_j, aux|\sum w_j C_j$ are described, for example, by Cakici and Mason [3]. Some metaheuristic methods have been used on unrelated parallel machine problems with sequence and machine dependent setup times. We proposed in this paper an integer LP with binary variables and a more detailed modeling taking into account mask management, batch-configuration, we have unrelated machines, sequence dependent setup times and different objective functions.

École Nationale Supérieure des Mines de Saint-Étienne - Centre microélectronique de Provence
E-mail: bitar@emse.fr dauzere-pere@emse.fr yugma@emse.fr

2 Problem description

We describe formally our problem in the following; we consider :

- A set $J = \{J_1, \ldots, J_n\}$ of jobs (or lots)and a set $M = \{M_1, \ldots, M_m\}$ of machines (or tools);
- A set $A = \{A_1, \ldots, A_\ell\}$ of auxiliary resources (or masks), necessary to process the jobs on machines.
- For each lot $J_i \in J$, the index of the mask $\varphi_i \in \{1, \ldots, \ell\}$ required for this lot.
- A set $\mathcal{M}_i \subset \mathcal{P}(\{1, \ldots, m\})$ giving the set of the qualified machines indices for each job J_i.
- For each job J_i, its priority as an integer c_i and its number of wafers w_i;
- For all $i \in \{1, \ldots, n\}$ and all $j \in \mathcal{M}_i$, the processing time $\rho_{ij} > 0$ of job J_i on machine M_j;
- For all J_i, $J_k \in J$, and all $j \in \mathcal{M}_i \cap \mathcal{M}_k$, the setup time β_{ijk} required by M_j to execute J_k immediately after J_i.
- For all $A_i \in A$ the initial location $R(A_i, 0) \in M \cup \{M_0\}$, i.e. the machine it is equipped on (or the storage if M_0).

The three objective functions considered are

- The weighted sum of the completion times to minimize.
- The number of moves, i.e. the number of wafers processed in the interval [0,H], to maximize.
- The number of mask changes, to minimize.

We can show that when the objective is the weighted sum of completion times, the problem is NP-hard, by polynomially reducing the NP-hard problem $R_m||\sum w_jC_j$ to it. We present here the sketch of the proof.

Let I be an instance of the problem $R_m||\sum w_jC_j$. We note $J_i, \ldots, J_n$ the jobs and $M_1, \ldots, M_m$ the machines. We have processing times ρ_{ij} and finally job weights w_j. Let π be our problem and I_π an instance of π, we have the following :

- n jobs $J_i, \ldots, J_n$.
- m machines $M_1, \ldots, M_m$.
- For all $i \in \{1, \ldots, n\}$, $\mathcal{M}_i = \{1, \ldots, m\}$ (no specific qualification).
- For all $i, i_0 \in \{1, \ldots, n\}$, $i \neq i_0$, $j \in \{1, \ldots, m\}$, $\beta_{iji_0} = 0$ (no setup times).
- Finally, $\ell = n$, $A = \{A_1, \ldots, A_n\}$, i.e. there is exactly one mask per job, so there's no couple of jobs requiring the same mask then no constraints concerning the auxiliary resource. Furthermore, we can consider that the initial location of every mask is M_0. Then, the first jobs on machines will begin at time $t = 1$.

Here, we see that we can polynomially reduce the problem π', which is $R_m||\sum w_jC_j$ where the first jobs on machines begin at time $t = 1$, to the problem π. We can then easily show that π' and $R_m||\sum w_jC_j$ are equivalent, because the cost c_s of every solution S of $R_m||\sum w_jC_j$ is the cost $c_s = c_{s'} - \sum_{j=1}^{n} w_j$, where $c_{s'}$ is the cost of an equivalent solution S' of π' consisting of beginning every lot one time unit later than in S. The cost translation is constant ($\sum_{j=1}^{n} w_j$ does not depend on the solutions) so finding the optimal solution for π' is equivalent to finding the optimal solution for $R_m||\sum w_jC_j$ (we can easily show it by contradiction).

These two steps we presented allow us to polynomially reduce the problem $R_m||\sum w_jC_j$ to π. Then, π is NP-Hard.

3 Linear programming formulation

The variables we use in our problem are the following :

- u_{ijt} for all $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, m\}$, $t \in \{0, \ldots, T - \rho_{ij}\}$.

$$u_{ijt} = \begin{cases} 1 \text{ if the job } J_i \text{ begins its processing at } t \text{ on } M_j, \\ 0 \text{ else.} \end{cases}$$

- a_{ijt} for all $i \in \{1, \ldots, \ell\}$, $j \in \{0, 1, \ldots, m+1\}$, $t \in \{0, \ldots, T\}$.

$$a_{ijt} = \begin{cases} 1 \text{ if mask } A_i \text{ is on } M_j \text{ at } t, \\ 0 \text{ else.} \end{cases}$$

When the objective function is not the number of mask changes, we can get rid of a_{ijt} variables. Then we have the following constraints :

1. $\forall i \in \{1, \ldots, n\}$

$$\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt} = 1.$$

This is to ensure that every job will only begin its processing once.

2. $E_j = \{i | j \in \mathcal{M}_i\}$. $\forall i \in \{1, \ldots, n\}, j \in \mathcal{M}_i$, $t \in \{0, \ldots, T - \rho_{ij}\}$

$$\sum_{\substack{i_0 \in E_j \\ i_0 \neq i}} \sum_{t_0=t}^{t+\rho_{ij}-1+\beta_{iji_0}} u_{i_0jt_0} \leq nT(1 - u_{ijt}).$$

These constraints express the fact that if a machine is processing a job J_i, no other job J_{i_0} will begin its processing on it until it finishes J_i, and we do not forget the β_{iji_0} setup time.

3. $\forall i \in \{1,\dots,n\}, j \in \mathcal{M}_i, t \in \{0,\dots,T-\rho_{ij}\}$

$$\sum_{\substack{io|\varphi_{i_0}=\varphi_i \\ i_0\neq i}} \sum_{\substack{j0\in\mathcal{M}_{i_0} \\ j\neq j_0}} \sum_{t_0=t}^{t+\rho_{ij}} u_{i_0jt_0} \leq mnT(1-u_{ijt}).$$

Here, we ensure the fact that for any couple of jobs requiring the same mask, there won't be a simultaneous processing on two different machines (unitary transport times of the masks are also considered).

4. for all $j \in \{1,\dots,m\}$,

$$\sum_{\substack{i\in E_j \\ R(\phi_i,0)\neq j}} u_{ij0} = 0.$$

We cannot begin the process of a lot when the required mask is not initially on the machine (at time $t = 0$).

With this modeling, the masks are implicitly taken into account. Then, we cannot count the number of mask moves this way. To optimize the number of mask moves we have to add the a_{ijt} variables and some constraints on it. These constraints are described in the following :

- For all $i \in \{1,\dots,\ell\}$, $t \in \{0,\dots,T\}$

$$\sum_{j=0}^{m+1} a_{ijt} = 1.$$

A mask can only be located at one place at a time.

- for all $i \in \{1,\dots,\ell\}$, r such that $r = R(A_i, 0)$

$$a_{ir0} = 1.$$

We know the initial locations of the masks.

- for all $i \in \{1,\dots,\ell\}$, $j \in \{1,\dots,m\}$, $t \in \{0,\dots,T\}$

$$\sum_{\substack{j_0=0 \\ j_0\neq j}}^{m} a_{ij_0(t+1)} \leq (m+1)(1-a_{ijt}).$$

Moving a mask requires a constant transport time (1).

- for all $i \in \{1,\dots,n\}$, $j \in \mathcal{M}_i$, $t \in \{0,\dots,T-\rho_{ij}\}$

$$\rho_{ij}u_{ijt} \leq \sum_{t_0=t}^{t+\rho_{ij}} a_{\varphi_i jt_0}.$$

A lot can only be processed on a machine holding the right mask.

Of course, in this version, we have to withdraw the constraints 3 and 4.

Some experimental results of our linear program will be provided during the conference.

4 Further work and conclusion

We have proposed an integer linear programming model to solve an unrelated parallel machine scheduling problem with auxiliary resources and setup times. Though, the formulation could be improved : it requires a large number of variables because it depends on time (time indices) so we have to be careful with the size of our instances. Moreover, there are many big M constraints. Our goal is now to think about a formulation without time indices.

Some heuristics and metaheuristics are often used for this type of problems and have revealed to be very efficient. We will also propose a metaheuristic method to provide good solutions for the problem.

More details and some experimental results on our formulation will be provided at the conference.

References

1. J.N.D., Gupta, R., Ruiz, J. W. Fowler, S. J. Mason (2006), Operational planning and control of semiconductor wafer production, *Production Planning and Control*, Vol. 17(7), 639 - 647.
2. S. L. M., de Diaz, J. W., Fowler, M. E., Pfund, G. T., Mackulak, M., Hickie, (2005), Evaluating the impacts of reticle requirements in semiconductor wafer fabrication, *IEEE Transactions on Semiconductor Manufacturing*, Vol. 18(4), 622 632.
3. E., Cakici, S. J., Mason, (2007). Parallel machine scheduling subject to auxiliary resource constraints, *Production Planning and Control*, Vol. 18(3), 217 225.

MISTA 2013

Grouping tasks to save energy in a cyclic wireless sensor network scheduling problem

Z. Hanzalek · C. Hanen

1 Motivation

In order to improve the functionality and efficiency of industrial monitoring and control systems, the industries are looking toward Wireless Sensor Networks (WSNs), especially in environments where electricity is not available. So, saving energy is an important tradeoff in such systems. We consider here scheduling of messages in the cluster-tree version of the leading standard IEEE 802.15.4/ZigBee[1][6].

A network of nodes in a tree-shape is considered, which leaves are sensors. The communications between nodes are given by a set of data flows through the network. The data flows repeat periodically. In order to save energy the Zigbee standard [6] specifies that each cluster is active only once during the period. Hence during a period, all tasks of data-flows passing through a cluster are grouped. Thus, the key problem is to find a periodic schedule which specifies when the clusters are active and transfer data on the links, while avoiding possible inter-cluster collisions and meeting all flows end-to-end deadlines, expressed by a bound on the number of periods needed to acheive the data flow.

This motivating problem, after a modeling step [5] led us in a previous paper [3] to address the more general problem setting where a cyclic dependence graph with uniform constraints is to be scheduled so that some tasks are grouped in order to save energy. In this paper, we investigate the special case where the structure of the cyclic precedence graph is reduced to a collection of independent cycles, and where the grouping follows a topological structure of a chain or a tree (like in Zigbee).

Zdenek Hanzalek
Czech Technical University in Prague, Faculty of Electrical Engineering
E-mail: hanzalek@fel.cvut.cz

Claire Hanen
LIP6, UPMC, Paris, and university of Paris-Ouest-Nanterre-La-Défense
E-mail: claire.hanen@lip6.fr

2 Problem setting

As we handle a cyclic scheduling problem with precedence and resource constraints, we use the denotations of [4]. The problem is defined by a set of tasks $\mathcal{T}$, assumed here to have unit processing times, to be repeated periodically. The tasks are linked by uniform precedence constraints, represented by a uniform graph G, with nodes $\mathcal{T}$. Each arc (u, v) of G has a nonnegative height (an integer value)h_{uv} that induces for any positive integer k a precedence constraint between occurrence k of u and occurrence $k + h_{uv}$of v. A periodic schedule of period λ assigns to each task u a non-negative integer starting time s_u so that

$$\forall (u, v)\ arc\ of\ G,\ s_v - s_u \geq 1 - \lambda h_{uv} \tag{1}$$

A periodic schedule can also be described by its core σ and its retiming α as follows: $\alpha(u)$ is the largest integer such that $s_u = \sigma(u) + \lambda\alpha(u)$, with $\sigma(u) \in [0, \lambda)$. The core represents the frame of the schedule during any period of the stationary state.

In addition, we consider that we are given a partition of tasks into groups $X_1, \ldots, X_m$. A periodic schedule is said to be grouped if the tasks of the same group are scheduled as a single task in the core schedule, so that the specific resources used by the tasks of a group have to be switched on only once in each period, and they perform all tasks either in sequence or in parallel, but grouped. So a schedule associates a starting time t_r to each group r in the core. Notice that as we address here feasibility questions, we can assume without loss of generality that the duration of each group is unitary (as if all tasks of a group were performed simultaneously). Indeed, the feasibility problem is to check whether there exists a periodic schedule (regardless of the period value). We proved in [3] that this problem is $NP-$complete, but in this paper, we adress a special case of the general problem where the groups and the precedence graph G have a particular topological structure.

In the Zigbee special case, we are given a tree T, whose nodes represent the types of tasks (groups) and whose edges represent communication links between them.

Each flow of data f induces a uniform graph G_f based on an oriented copy of a subtree of T, oriented as an in-tree. If i is the underlying node of T then the copy of i in G_f is denoted by u_f^i. The in-tree represents the communication of data along the communication links of T from source nodes of the flow to the unique sink $\bar{u}_f^j$. Moreover, the expected quality of service is expressed by definining, for each source node u_f^i an arc from u_f^i to $\bar{u}_f^j$ with height h_f wich represents the maximum number of periods allowed to deliver the data.

The uniform graph G associated to the flows is just the juxtaposition of the flow subgraphs.

The groups are associated to the nodes of the underlying communication tree T: for each node i of T, there is a group X_i to which belong all nodes of the form u_f^i.

We denote by $ZIG - TREE$ the feasibility problem in this case , and by $ZIG - CHAIN$ the feasibility problem when the tree T is a chain.

3 Study of $ZIG - CHAIN$ and $ZIG - TREE$

We first consider the special case where the groups are organized along a chain. Despite the complexity of checking feasibility in this subcase is still unknown, we show

that a linear program solvable in polynomial time using graph algorithms provides a necessary feasibility condition. Moreover, by enforcing constraints, we provide another sufficient condition. From these properties we derive an integer linear program that models feasibility. We then consider an optimization criteria related to the quality of a solution and we show that the techniques developed previously lead to a polynomial heuristic.

When the underlying structure is a chain, G is reduced to a collection of circuits, one for each flow. Moreover, we can distinguish "'increasing flows" (flows of data from left to right of the chain) and "'decreasing flows" running in the opposite direction.

Our work first express the problem as a graph problem, based on the retiming approaches [2]

Property 1 *' The problem has a feasible solution if and only if one can remove at most h_f arcs of each flow circuit G_f and merge the nodes of the same groups without creating a cycle in the resulting graph.*

This property lead us to observe that for any communication link between two consecutive nodes $[k, k+1]$ all remaining arcs crossing this link will cross it in only one direction (otherwise a cycle would appear). This remark lead us define new variables N_j associated to each group j that denotes the number of links $[k, k+1]$ with $k+1 \leq j$ crossed from left to right.

We can now state a necessary feasibility condition, and a sufficient one based on these variables.

Theorem 1 *For any feasible solution, the associated variables N_j satisfy the following inequalities:*

$$\begin{cases} \forall k \in [0, n-2] & 0 \leq N_{k+1} - N_k \leq 1 \\ r_f - s_f - (N_{r_f} - N_{s_f}) \leq h_f & for\ any\ increasing\ flow\ f \\ (N_{s_f} - N_{r_f}) \leq h_f & for\ any\ decreasing\ flow\ f \end{cases} \quad (2)$$

Conversely, if one of the two following inequation system has a solution then the problem is feasible:

$$\begin{cases} \forall k \in [0, n-2] & 0 \leq N_{k+1} - N_k \leq 1 \\ r_f - s_f - (N_{r_f} - N_{s_f}) \leq h_f - 1 & for\ any\ increasing\ flow\ f \\ (1 \leq N_{s_f} - N_{r_f}) \leq h_f & for\ any\ decreasing\ flow\ f \end{cases} \quad (3)$$

$$\begin{cases} \forall k \in [0, n-2] & 0 \leq N_{k+1} - N_k \leq 1 \\ 1 \leq r_f - s_f - (N_{r_f} - N_{s_f}) \leq h_f & for\ any\ increasing\ flow\ f \\ (N_{s_f} - N_{r_f}) \leq h_f - 1 & for\ any\ decreasing\ flow\ f \end{cases} \quad (4)$$

By combining the two conditions, we propose an integer linear program on the variables N_j and boolean decision variables x_f for each flow f that models the feasibility problem.

Theorem 2 *In the Zigbee-chain problem, the feasibility can be modeled with the following ILP:*

$$\begin{cases} for\ any\ increasing\ flow\ f & x_f \leq r_f - s_f - (N_{r_f} - N_{s_f}) \leq h_f + 1 - x_f \\ for\ any\ decreasing\ flow\ f & x_f \leq N_{s_f} - N_{r_f} \leq h_f + 1 - x_f \\ \forall k \in [0, n-2], & v_{k+1} - v_k \geq (M+1)(N_{k+1} - N_k) - M \\ \forall k \in [0, n-2], & v_k - v_{k+1} \geq 1 - (M+1)(N_{k+1} - N_k) \\ \forall\ flow\ f & x_f \in \{0, 1\} \end{cases} \quad (5)$$

We then introduce some flexibility in the model by considering variable parts in the heights of the circuits modelling the data flows, in order to provide solutions in any cases, minimizing the deviation with respect to the orignal quality requirements either for a fixed period or for any period. We show that the previous integer program can be easily extended to handle such optimization criteria. Moreover, our sufficient condition of theorem 1 lead to a polynomial algorithm to compute an approximated solution.

Finally, we investigate the generalization of the results for the $ZIG-TREE$ problem. We show that a necessary condition that can be checked in polynomial time can also be stated and we propose an integer linear program to model the problem.

To conclude, although the complexity of $ZIG-CHAIN$ and $ZIG-TREE$ remain unknown, our application of tools coming from the cyclic scheduling with precedence constraints on this special case leads to a new insight on such problems, and to promising algorithmic approaches.

References

1. IEEE SA Standards Board. Part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs), 2006.
2. Alain Darte and Guillaume Huard. Loop shifting for loop compaction. *Lecture Notes in Computer Science*, 1863/2000:415–431, 2000.
3. Claire Hanen and Zdenek Hanzalek. Grouping tasks to save energy in a cyclic scheduling problem: a complexity study. In *ROADEF 2012*, April 2012.
4. Claire Hanen and Alix Munier. Cyclic scheduling on parallel processors: An overview. In Philippe Chrétienne, Edward G. Coffman, Jan Karel Lenstra, and Zhen Liu, editors, *Scheduling theory and its applications*. J. Wiley and sons, 1994.
5. Z. Hanzalek and P. Jurcik. Energy efficient scheduling for cluster-tree wireless sensor networks with time-bounded data flows: Application to ieee 802.15.4/zigbee. *Industrial Informatics, IEEE Transactions on*, 6(3):438 –450, august 2010.
6. ZigBee Standards Org. Zigbee specification, 2006.

MISTA 2013

A 3-phase approach to geographically distributed exam timetabling

M. Mihaylov · T. Wauters · G. Vanden Berghe

1 Introduction

Exam timetabling is a well-studied domain where a large number of student exams need to be scheduled in a short period of time, taking into account a number of constraints (Qu et al, 2009). These constraints may limit the number of exams in a day, or may spread exams evenly for any one student. The literature on exam timetabling mainly assumes that examinations take place at a fixed location (e.g. in a school, or in a university campus) (Gaspero and Schaerf, 2001), or different campuses of the same university (Kaplansky et al, 2004). However, some real-world problems exist that require the scheduling of exams based on the geographical distribution of students (henceforth called candidates or participants), in locations that are not a priori fixed. For example, a certifying organization may want to conduct certification tests in multiple cities as close as possible to all candidates, while taking into account room availability and costs, as well as the travel distance, availability and fees of the exam supervisors. It is therefore our belief that the literature on exam timetabling needs to be extended to cover such complex real-world geographically distributed scenarios. Here we present the problem of geographically distributed exam timetabling, which to the best of our knowledge has not been addressed in the literature.

A large number of candidates from different cities enroll for certification exams, such as for Cisco certificates or certificates for Microsoft products. Each candidate specifies his/her preferred exam location in a web-based platform. Depending on the number of participants, a date is chosen by the organizer (or planner) and suitable locations are selected, based on the location preferences of candidates. An exam can take place on multiple locations, but has to be carried out at the same time, which is determined based on room availability. Thereafter, an exam supervisor is selected for each room.

This problem has been identified by Televic Education[1], which intends to offer a tool that supports the human planner at certifying organizations in this challenging

Mihail Mihaylov, Tony Wauters, Greet Vanden Berghe
CODeS, KAHO Sint-Lieven, Departement of Computer Science, Gent, Belgium
E-mail: mihail.mihaylov@kahosl.be, tony.wauters@kahosl.be, greet.vandenberghe@kahosl.be

[1] www.televic-education.com

task. Here we introduce an approach for addressing the above problem, which will later become part of a decision support tool.

2 A decomposition approach

We decompose the geographically distributed exam timetabling problem and propose a 3-phase approach to address it, where assignments are solved sequentially – the output of each phase is used by the following one. The proposed approach first *finds locations* that minimize the travel distance of all participants, then determines the *schedules of exams* that minimize the room occupancy at that location in order to reduce the costs for the organization, and finally minimizes the cost for *hiring exam supervisors*. Below we elaborate on each phase in this optimization process.

Phase 1: Location assignment. To minimize the travel distance of all candidates, one needs to find the geographical center of all location preferences. As candidates can be located in multiple cities, finding multiple geographical centers can further reduce the distance to all participants, at the expense of higher organizational costs. The course organizer determines the maximum number of locations k over which each exam e can be spread and the required minimum number of participants p at any given location. The grouping of participants is performed with the help of k-means clustering, where the number of clusters for each exam is k. The initial centroid of each cluster is set as the location preference of one uniformly randomly selected candidate. In other words, each data point to be clustered represents the preferred location of one candidate on the map. At each iteration the clustering algorithm computes the closest centroid for each data point and then moves each centroid to the geographical center of all data points in that cluster. This process is repeated until the locations of all centroids do not change more than a small number ϵ between two iterations. Without loss of generality, we chose to use euclidean distances, instead of actual road distances. After determining the geographical center of each cluster, an exam location is selected that is the closest to each centroid. Clusters with less than the minimum number of participants p for an exam are re-assigned to the closest location that has p or more participants. Alternatively, several clusters with less than p participants can be joined. The final number of locations for each exam may be k or lower, since multiple centroids can be close to the same exam location, and because small clusters may be merged with larger ones.

A number of different solutions with comparable average travel distance can be generated, due to the stochastic nature of the clustering algorithm. While k-means optimizes for the average travel distance, additional fairness metrics and organizational preferences can be considered when deciding on the final assignment of locations. Examples of additional factors that can influence the choice of a solution are the maximum travel distance, the final number of locations per exam, and the use of any particular (un)desired locations.

Phase 2: Room assignment. Once locations are determined based on the above metrics, exams at each location need to be assigned to available rooms at that location, taking into account the size of the exam, its equipment requirements and the occupancy schedule of the room. Our approach also takes into account all typical constraints in examination timetabling, such as that a participant can only take one exam

at a time, a given exam taking place in multiple locations has to be carried out at the same time (Kaplansky et al, 2004), etc. Each exam location may contain multiple rooms r of different capacity r_c, where in some rooms certain equipment q is present, e.g. computers, a projector, etc. Each exam e may require equipment e_q to be present in the room. Thus, exams that require equipment e_q may only be scheduled in rooms with equipment q. In contrast, exams with no equipment requirement can be scheduled in any room, regardless of the available equipment. As some rooms may be reserved for other activities or be otherwise unavailable at given time slots (e.g. due to maintenance), such unavailabilities need to be taken into account in the assignment process. We use a general purpose MILP solver to find an allocation of exams to rooms that minimizes the sum of capacities r_c of all rooms used for exams, given the above constraints. The intuition behind this objective is that larger rooms have higher reservation costs. Starting times of exams are also determined in the allocation process.

The MILP solver yields the optimal allocation of exams to rooms, or a near-optimal one, in case computation time exceeds some pre-defined threshold. Thereafter, exam location, room, date and time are fixed and can be communicated to all participants. Once exam times are fixed, one can still accommodate any late changes to room schedules. Assigning a set of exams with fixed starting times to a set of rooms draws parallels to the shift minimization personnel task scheduling problem (Krishnamoorthy et al, 2012). Approaches, such as that proposed in (Smet and Vanden Berghe, 2012), for solving the latter problem can be applied for minimizing the number of used rooms.

Phase 3: Supervisor assignment. Lastly, having the exam locations and schedules, an exam supervisor s is assigned to each exam. Among all supervisors available at the time of each exam, our approach selects those that minimize the weighted sum of the fees s_f of that supervisor and his/her distance s_d to that respective location, where s_f is being weighted 5 times more than s_d. In this way the organization will hire an affordable supervisor, located relatively close to the exam location.

3 Summary and outlook

Geographically distributed exam timetabling is a challenging task that some organizations are often confronted with. To address this problem, we introduced a 3-phase approach, which can be integrated in a decision support tool that guides the organizer at different stages of the planning process. Our approach integrates individual preferences of candidates, and adequately balances the cost-quality trade-off of solutions. Each generated solution can provide information to the planner on the overall cost of the allocation, i.e. room and supervisor costs, as well as the average and maximum travel distance of participants. These measures allow planners to compare the different solutions and select the one that best fits the objectives of the organization.

In the proposed approach we tackle the distributed exam timetabling problem a sequential manner. In the first phase locations are fixed that are close to all participants, then exams are assigned to rooms in those locations and finally supervisors are selected for each exam. In future work we will consider an iterative process where each assignment uses the output of the others, further optimizing the costs of the organization. For example, room assignment should be taken into account in the clustering algorithm; large exams can be split in several smaller rooms; the availability of supervisors should also be considered when determining the starting times of exams; etc. In addition, the

parameter k, or the maximum number of locations per exam, can be automatically determined by generating solutions with different values for k and selecting the result with the best cost-quality trade-off, as defined by the organization.

Acknowledgements This research was carried out within the IWT 110739 project (Geoxam) with Televic Education.

References

Gaspero L, Schaerf A (2001) Tabu search techniques for examination timetabling. In: Burke E, Erben W (eds) Practice and Theory of Automated Timetabling III, Lecture Notes in Computer Science, vol 2079, Springer Berlin Heidelberg, pp 104–117

Kaplansky E, Kendall G, Meisels A, NaimahHussin (2004) Distributed examination timetabling. In: In Proc. of the 5th International Conference of the Practice andTheory of Automated Timetabling (PATAT), pp 511–516

Krishnamoorthy M, Ernst A, Baatar D (2012) Algorithms for large scale shift minimisation personnel task scheduling problems. European Journal of Operational Research 219:34–48

Qu R, Burke E, McCollum B, Merlot L, Lee S (2009) A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12:55–89

Smet P, Vanden Berghe G (2012) A matheuristic approach to the shift minimisation personnel task scheduling problem. In: Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT), Son, Norway, pp 145–160

MISTA 2013

The Balanced Path Problem in Crew Scheduling Applications

Güvenç Şahin • Fardin Dashty Saridarq

1 Introduction

We study the operational level planning of crew resources. The assignment of crew members to a sequence of duties in a finite planning horizon is determined in such a way that the allocation of the workload among the crew members is acceptably fair and almost equal when possible. This issue is a common planning phenomenon for also other type of duty scheduling and rostering processes where teams of crew members are in consideration such as hospitals and airlines. At the tactical level, the crew schedules are feasible with respect to various restrictions and regulations; yet, they may result in an imbalanced share of workloads among the crew. In addition, unbalanced crew schedules may also cause unavoidable over-time costs and result in unevenness with respect to time-based compensations.

We develop both exact and heuristic approaches to solve the workload balancing problem for a given crew team. As the exact approach, we formulate a network flow problem considering all rules and restriction. Two classes of heuristic methods have been developed:

- A two-phase approach solves an integer programming formulation at each phase.
- With a pool of conventional operators, we develop a heuristic framework that modifies the schedules by reallocating the duties without disrupting the feasibilities.

We present the results of our computational experiments with a data set that is representative of one of the largest crew regions in Turkish State Railways.

2 Problem Description and Exact Mathematical Formulation

Number of studies concerned with the balanced path problem in crew scheduling literature is limited. Existing studies on the balanced path problem do not address the issues in scheduling applications. There are some additional constraints due to the nature of scheduling problems that should be considered in the workload balancing context.

We use the network representation of the crew scheduling problem described in Şahin and Yüceoğlu [2]. According to this network $G = (V, A)$, a pair of on-duty node and a tie-up node is representing the beginning and end of a duty, respectively, with location and time information. A duty is represented by a duty arc $(i, j) \in A$ emanating from an on duty node $i \in V$ and enetering a tie-up node $j \in V$; a path representing a feasible sequence of duties includes duty arcs as well as other types of arcs representing rest periods, duty connections, and off-days. Based on this network representation, the workload balancing problem finds a set of balanced paths.

In our network flow formulation of the problem, $x_{ij}^h = 1$ if arc (i, j) is included in the feasible path h; otherwise, $x_{ij}^h = 0$. When (i, j) is included in a path, it means that it is covered by the schedule represented by the path. W_{ij} denotes the workload of duty arc (i, j) ; it is zero for all arcs other than duty and deadhead arcs. For a duty arc (i, j), C_{ij} is the number

Güvenç Şahin
Manufacturing Systems and Industrial Engineering, Sabanci University, Istanbul, Turkey
E-mail: guvencs@sabanciuniv.edu

Fardin Dashty Saridarq
Industrial Engineering, Sabanci University, Istanbul, Turkey
E-mail: fardin@sabanciuniv.edu

of crew members required to cover the duty represented by arc (i, j). Then, the network flow formulation for the workload balancing problem becomes

$$\min \quad Z_{max} - Z_{min} \tag{1}$$

$$\text{subject to} \sum_{(s,i)\in A} x_{si}^h = 1 \qquad h = 1, \dots, k \tag{2}$$

$$\sum_{(i,t)\in A} x_{it}^h = 1 \qquad h = 1, \dots, k \tag{3}$$

$$\sum_{(j,i)\in A} x_{ji}^h - \sum_{(i,j)\in A} x_{ij}^h = 0 \qquad h = 1, \dots, k \quad \forall i \in V \backslash \{s, t\} \tag{4}$$

$$\sum_{h} x_{ij}^h \geq C_{ij} \qquad \forall (i, j) \in A \tag{5}$$

$$\sum_{(i,j)\in A} W_{ij} x_{ij}^h \leq Z_{max} \qquad h = 1, \dots, k \tag{6}$$

$$\sum_{(i,j)\in A} W_{ij} x_{ij}^h \geq Z_{min} \qquad h = 1, \dots, k \tag{7}$$

$$\sum_{h} \sum_{(i,j)\in A} W_{ij} x_{ij}^h \leq W_T + \epsilon \tag{8}$$

$$x_{ij}^h \geq 0 \qquad h = 1, \dots, k \;\; \forall (i, j) \in A \tag{9}$$

where W_T is the minimum total workload to cover all duties in the planning horizion with the minimum number of required crew members. The objective function (1) minimizes the workload difference between the two crew schedules (i.e. feasible paths) where Z_{max} and Z_{min} are decision variables denoting the maximum and the minimum workload on the schedules. Constraints (2), (3) and (4) are flow conservation constraints which assure that, for each path h, a unitary flow is pushed from the source node to the sink node, representing the engagement of only one crew with the feasible schedule represented by the path. Constraint (5) assures that, all duties are covered. Constraints (6) and (7) ensure that all paths have a workload in the range of the minimum and the maximum workload, and also determine the value of the workload for the maximum-workload path and the minimum-workload path, respectively. Constraint (8) assures that the total workload of all paths does not exceed the minimum total workload and constraint (9) is the domain constraint for the decision variables.

The proposed network flow formulation finds an optimal solution for the workload balancing problem. However, in order to decrease the computational effort required to solve the problem some improvements can be done in order to reduce the feasible region of the problem formulation. In addition to a feasible upper bound on the total workload, we also impose a lower bound as $\sum_h \sum_{(i,j)\in A} W_{ij} x_{ij}^h \geq W_D + \epsilon$ where the lower bound W_D can be found as the summation of workload of all duties to be covered within the planning horizon.

3 Heuristic Approaches

3.1 Two-Phase Integer Programming

We modify the exact formulation given in (1)-(9) with some changes in constraints and solve the overall problem in two phases. This approach reduces the complexity of the problem, but provides a suboptimal solution as a heuristic procedure. At the first phase, we solve the following problem [P1]:

$$\min \quad \bar{Z}_{min} \tag{10}$$

subject to $(2)-(5),(8)-(9)$

$$\sum_{(i,j)\in A} W_{ij}x_{ij}^{h} - M(1-y^{h}) \le \bar{Z}_{min} \qquad h = 1,\dots,k \tag{11}$$

$$\sum_{(i,j)\in A} W_{ij}x_{ij}^{h} \ge \bar{Z}_{min} \qquad h = 1,\dots,k \tag{12}$$

$$\sum_{h} y^{h} \ge 1 \tag{13}$$

$$y^{h} \in \{0,1\} \qquad h = 1,\dots,k \tag{14}$$

where $y^h = 1$ if the corresponding path is the feasible schedule with the minimum possible workload. The objective function (11) minimum possible workload $\bar{Z}_{min}$. Constraints (11), (12) and (13) ensure that there is at least one path with workload amount of $\bar{Z}_{min}$. Constraint (14) is the binary domain constraint.

$\bar{Z}^{*}_{min}$ denoting the value of the optimal objective function value of [P1] provides a lower bound for Z^{*}_{min} in the optimal solution of $(1)-(9)$, i.e. $\bar{Z}^{*}_{min} \le Z^{*}_{min}$. Then, the following problem [P2] finds the minimum of the maximum workload of all paths:

$$\min \quad \bar{Z}_{max} \tag{15}$$

subject to $(2)-(5),(8)-(9)$

$$\sum_{(i,j)\in A} W_{ij}x_{ij}^{h} \ge Z_{\min} \qquad h = 1,\dots,k \tag{16}$$

If $\bar{Z}^{*}_{max}$ denotes the optimal objective function value, then $\bar{Z}^{*}_{max} - \bar{Z}^{*}_{min}$ provides an upper bound for the original workload balancing problem in (1)-(9).

3.2 Conventional Heuristic Methods

Our heuristic idea improves the difference between the two schedules with maximum-workload and the minimum workload by exchanging some duties between two schedules while considering all constraints and rules. Given a pair of schedules, two operators work in the following manner to generate two new schedules:

- Crisscross operator breaks each schedules into two parts in such a way that the early part of the first schedule can be followed by the later part of the second schedule while the the early part of the second can be followed by the later part of the first.
- The swap operators finds one duty in each of the schedules and replaces these duties into the other schedule.

The heuristic algorithm works in an iterative manner; at each iteration two schedules are selected, and the operator is implemented on the selected pair if the workload difference between the two schedules is to decrease after the operator is implemented. The pairs are selected from the sorted list of schedules in an increasing workload order. The algorithm iterates as follows:

- At the first iteration, the first (minimum-workload) and the last (maximum workload) schedules are selected.
- If the workload difference is improved, the list is updated; the first and the last schedules in the updated list are selected again. Otherwise, the first schedule and the one before the last schedule are selected.
- The pair selection proceeds from the bottom of the list until the first and the second are selected. If no improvement is attained, the second schedule is paired with the last one.

The algorithm continues in this fashion until the first schedule of the selected pair is in the middle of the list.

4 Computational Study

Our computational experiments are conducted with a set of problem instances from Turkish Railways. The solution methods are implemented using C++ and CPLEX Concert Technology. Table 1 shows the results for one such instance from the problem set. The best results are obtained with a hybrid implementation of the conventional heuristic algorithm where each iteration implements the change imposed by the operator leading to a better improvement for the selected pair. When the integer programming formulations are used, the optimality could not be attained after long hours of computation. We present the upper bound values attained after 30 hours.

	OFV	CPU Time
Exact Algorithm	30.1 hours	30 hours
Exact Algorithm-Using Lower Bound	8.86 hours	30 hours
Two-Phase IP	7.96 hours	30 hours
Crisscross	8.08 hours	< 1 min
Swap	12.05 hours	< 1 min
Hybrid	6.95 hours	< 1 min

Table 1-Results for exact and heuristic methods

The initial maximum workload difference on the original schedule for this instance is 21.283 hours. As the imbalance at the tactical level can be reduced by more than 50%, we believe that the heuristic results are extremely satisfactory from both practical and algorithmic points of view.

Acknowledgements This research has been supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under Grant 110M495.

References

1. Cappanera, P. and Scutellà, M., Color-Coding Algorithms to the Balanced Path Problem: Computational Issues, INFORMS Journal on Computing 23, 446-459 (2011).
2. Şahin, G. and Yüceoğlu, B., Tactical Crew Planning in Railways, Transportation Research Part E: Logistics and Transportation Review 47, 1221–1243 (2011).

MISTA 2013

Parallel Machine Scheduling with Generalized Precedence Relations

Jinil Han · Ruslan Sadykov · François Vanderbeck

1 Introduction

Parallel machine scheduling entails assigning jobs to identical machines, $k \in \{1, \ldots, m\}$, and deciding on their start time. Each job $j \in \{1, \ldots, n\}$ requires a given processing time p_j. Its starting time is denoted by S_j (its completion time is then $C_j = S_j + p_j$). The objective is typically some regular function of the completion times. Here, we additionally consider generalized precedence relations between jobs, which can be represented as

$$S_j \geq S_i + l_{ij}, \tag{1}$$

where l_{ij} is the lag between job i and j which can be positive or negative. This constraint can be used to model various types of temporal dependencies between jobs such as the requirement of synchronization, precedence, and overlap between jobs by assigning appropriate values to l_{ij}'s. The set Δ defines all job pairs (i, j) for which a generalized precedence relation exists.

For this problem, the time-indexed formulation introduced in [1] can be extended to include (1):

$$[\text{P}] \quad \min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}, \tag{2}$$

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1, \qquad j = 1, \ldots, n, \tag{3}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq m, \qquad t = 1, \ldots, T, \tag{4}$$

$$\sum_{s=t-l_{ij}+1}^{T-p_i+1} x_{is} + \sum_{s=1}^{t} x_{js} \leq 1, \qquad (i,j) \in \Delta, t = 1, \ldots, T - p_j + 1, \tag{5}$$

$$x_{jt} \in \{0, 1\}, \qquad j = 1, \ldots, n, t = 1, \ldots, T - p_j + 1, \tag{6}$$

INRIA team RealOpt and Mathematics Institute, University of Bordeaux, France
E-mail: jinil.han@inria.fr, ruslan.sadykov@inria.fr and fv@math.u-bordeaux1.fr

where the binary variable $x_{jt} = 1$ if job j is to start in period t, and 0 otherwise. Note that constraint (5) is one of the strongest way to express precedence relations between jobs, but it entails a large number of constraints. Moreover, by replacing constraint (4) with the following flow conservation constraints we can get a variant which is known to be computationally more efficient [2]:

$$\sum_{j=1}^{n} x_{j1} + y_1 = m, \tag{7}$$

$$\sum_{j=1}^{n} (x_{jt} - x_{j,t-p_j}) + y_t - y_{t-1} = 0, \qquad t = 2, \ldots, T \tag{8}$$

$$y_t \in \mathbb{Z}_+, \qquad t = 1, \ldots, T \tag{9}$$

where the integer variable y_t indicates the number of idle machines in period t. Let [F] be the formulation defined by (2)-(3) and (5)-(9). Its feasible solutions correspond to flows with value m in a network N whose nodes represent periods and where a job j is represented by arcs $(t, t + p_j)$, and idle times by arcs $(t, t + 1)$. The important advantage time-indexed formulations [P] and [F] is that they provide tight LP bounds. Their drawback is the large number of variables, especially when processing times become large.

2 Column-and-row generation approach

One possible approach to overcome the large size of [P] and [F] is to handle the variables and constraints dynamically by applying a column generation paradigm to these extended formulations [3]. In this approach, constraints (3) and (5) form the master problem, and the flow constraints (7)-(8) go to pricing subproblem. For each machine, the pricing subproblem amounts to generating a pseudo-schedule, i.e., a schedule in which jobs can be processed more than once. Solving the pricing subproblem corresponds to finding the shortest path in acyclic network N. Then, each generated pseudo-schedule is split into individual x and y variables, which are then added to the master. Once the LP relaxation of the master is solved, the most violated precedence constraints are added to it, and the process is repeated. The procedure terminates when no improving columns (variables) and no violated constraints are found. The reasoning behind adding each variable by splitting pseudo-schedule unlike standard column generation is to allow for the recombination of previously generated pricing problem solutions, and thus to accelerate the convergence of column generation. For a detailed explanation, see [3], where parallel machine scheduling is one of the test problem. However, here, the presence of the large number of additional precedence constraints reduces the great performance of the algorithm compared to a direct MIP-solver approach for [P] or [F].

3 Preliminary computational results

We performed the preliminary computational tests to see how well a column-and-row generation approach is in comparison with other four approaches: solving [P] directly using CPLEX(`[P]-d`); solving [F] directly using CPLEX(`[F]-d`); solving [P] and [F]

by cut generation approach based on CPLEX where the precedence constraints being generated as cuts, denoted respectively (`[P]-c`) and (`[F]-c`). Column-and-row generation algorithm was implemented using BapCod, a generic Branch-and-Price code developed by the INRIA RealOpt team in Bordeaux, and all LP problems were solved using CPLEX 12.4. All tests were performed on Intel 2.50 GHz PC with 4GB RAM.

Table 1 shows the performance comparison between our column-and-row generation approach and other approaches on instances with 25 jobs and a total weighted tardiness objective function. Test instances were generated using the procedure from [4] which is the most used in the literature. We report average results over 5 instances with 25 jobs since most of instances with 50 jobs cannot be solved within a hour. We consider single and two machine cases. Instance classes (given in the column `num`) vary by the parameters RDD and TF . For each pair of job i and j such that $i < j$, the precedence constraint are randomly imposed with probability `p` in percent. We only considered the case of $l_{ij} = p_i$.

The headings of Table 1: `%var`, `%cut`, `#itr`, `mast`, `sub`, and `total`. They refer respectively to the percentage of generated variables, the percentage of generated constraints of type (5), the number of iterations, the time spent in solving master problems (in seconds), the time spent in solving pricing subproblems, and the total solution time, respectively. For direct cplex approach and cut generation approach, we only report total time. The fastest approach among the five is highlighted in bold type.

The case where `%p` $= 0$ represents the standard parallel machine scheduling problem without any precedence constraint. When the number of precedence relations gets larger, the number of variables and constraints generated increases rapidly, which leads to reducing the performance of column-and-row generation approach compared to CPLEX. Indeed, as the master becomes large, the column-and-row generation approach spent a great deal of time solving it at each iteration. We can also observe that the column-and-row generation approach seems to perform better compared to other approaches for most of tested instances, whereas the CPLEX solver performs better for several instances for which the relatively large number of precedence constraints are generated.

In other words, when there are few precedence constraints, the dynamic approach is performing better than solving the formulation directly with a MIP solver, or than only handling precedence constraints dynamically. But the reverse can sometimes be observed when the precedence graph is dense. Our future research plan is to examine special precedence graph as those arising in applications with synchronization constraints.

References

1. M.E Dyer, L.A. Wolsey, Formulating the single machine sequencing problem with release dates as a mixed integer problem, Discrete Applied Mathematics, vol.26, pp.255-270 (1990)
2. Y. Pan, L. Shi, On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems, Mathematical Programming, vol.110, pp.543-559 (2007)
3. R. Sadykov, F. Vanderbeck, Column Generation for Extended Formulations, EURO Journal on Computational Optimization, vol.1, pp.81-115 (2013)
4. C.N. Potts, L.N. Van Wassenhove, A branch and bound algorithm for the total weighted tardiness problem, Operations Research, vol.33, pp.363-377 (1985)

Table 1 Computational results

Instance				column-and-row generation						CPLEX			
m	n	num	%p	%var	%cut	#itr	mast	sub	total	[P]-d	[P]-c	[F]-d	[F]-c
1	25	1	0	5.0			0.3	0.1	**0.4**	8.5		23.1	
			1	4.6	0.2	8	0.5	0.1	**0.7**	1630.8	119.3	>1h	256.5
			3	5.2	0.3	11	2.0	0.1	**2.2**	>1h	>1h	>1h	1452.7
			5	3.8	0.3	11	0.6	0.1	**0.9**	>1h	>1h	>1h	>1h
			10	10.7	0.9	53	155.6	0.2	**156.7**	>1h	>1h	>1h	>1h
1	25	7	0	4.1			0.2	0.1	**0.3**	16.8		20.5	
			1	16.7	4.7	100	167.9	1.1	**171.0**	452.3	218.6	>1h	598.0
			3						>1h	>1h	>1h	>1h	>1h
			5						>1h	>1h	>1h	>1h	>1h
			10						>1h	>1h	>1h	>1h	>1h
1	25	13	0	5.3			0.5	0.1	**0.8**	9.5		22.4	
			1	4.5	0.1	12	0.9	0.2	**1.4**	57.9	25.0	471.1	30.9
			3	9.1	1.6	104	27.3	1.0	**29.5**	217.9	122.0	2213.9	166.9
			5						>1h	>1h	>1h	>1h	>1h
			10						>1h	>1h	>1h	>1h	>1h
1	25	19	0	6.6			0.3	0.1	**0.5**	9.7		20.6	
			1	6.1	0.7	53	2.9	0.4	**3.6**	52.9	21.2	615.0	47.1
			3	10.0	2.0	115	16.1	0.6	**18.4**	249.0	101.5	1159.4	240.8
			5	18.6	5.8	178	665.8	3.0	671.5	**587.5**	1030.4	>1h	1055.4
			10	21.8	4.5	204	1527.9	4.6	**1543.1**	>1h	>1h	>1h	>1h
1	25	25	0	6.2			0.7	0.1	**1.0**	35.8		45.5	
			1	5.5	0.0	1	4.5	0.3	**5.0**	125.0	77.5	1141.4	143.9
			3	4.6	0.0	1	0.4	0.1	**0.6**	128.6	276.9	1159.0	455.0
			5	10.7	1.9	79	70.3	1.2	**73.9**	1186.5	1672.6	>1h	2188.9
			10	21.0	3.9	177	2082.1	4.8	**2096.6**	>1h	>1h	>1h	>1h
2	25	1	0	5.6			0.1	0.0	**0.2**	7.5		3.6	
			1	6.9	0.8	13	1.0	0.1	**1.2**	116.6	24.3	1133.9	41.8
			3	9.4	0.8	17	5.2	0.9	**5.5**	482.5	114.7	>1h	1462.4
			5	16.4	3.8	76	125.1	0.6	**126.7**	1113.1	2493.4	>1h	1840.4
			10						>1h	>1h	>1h	>1h	>1h
2	25	7	0	3.3			0.0	0.1	**0.2**	5.2		4.3	
			1	19.2	13.7	168	242.3	4.6	254.7	158.6	**41.7**	1350.4	528.5
			3	23.2	11.7	176	1265.8	9.0	1288.1	238.2	**160.3**	1796.2	844.6
			5	25.6	10.0	191	2441.5	14.4	2471.1	**527.3**	553.8	>1h	3397.2
			10						>1h	>1h	>1h	>1h	>1h
2	25	13	0	5.5			0.2	0.1	**0.4**	4.3		4.0	
			1	9.1	3.0	58	2.2	0.2	**3.0**	26.0	10.1	424.7	9.7
			3	10.7	2.8	84	6.5	0.7	**8.1**	49.6	15.0	117.9	15.9
			5	17.0	3.6	92	25.5	0.9	**28.4**	97.9	102.2	316.7	147.8
			10	12.8	1.9	51	10.8	0.6	**13.1**	137.6	175.2	445.4	272.1
2	25	19	0	5.0			0.1	0.1	**0.3**	2.5		3.1	
			1	7.2	0.6	12	0.5	0.1	**0.8**	22.5	9.8	39.6	7.5
			3	10.1	1.1	60	1.8	0.1	**2.5**	40.7	11.6	96.1	14.5
			5	16.3	5.8	112	35.8	0.8	**39.3**	133.1	156.3	423.2	216.8
			10	16.0	3.7	92	33.0	1.0	**36.6**	254.6	277.4	611.7	363.2
2	25	25	0	4.1			0.2	0.0	**0.3**	18.3		6.0	
			1	6.2	0.7	13	0.6	0.1	**0.8**	31.5	38.3	107.5	14.8
			3	8.3	1.4	62	2.8	0.2	**3.6**	47.8	54.9	242.3	44.5
			5	11.3	1.7	35	3.8	0.6	**5.3**	132.2	179.1	523.9	314.1
			10	13.8	2.5	55	27.1	0.8	**29.6**	250.6	457.8	952.3	1459.3

MISTA 2013

Building university course timetables with minimized resulting student flows

A case study at the Hogeschool-Universiteit Brussel

Jeroen Beliën • Annelien Mercy

1 Introduction

The growing student numbers at colleges and universities have resulted into an enlarged complexity in terms of planning and organization. One of the tasks that becomes increasingly complex is the development of course timetables. Course timetables have to satisfy various requirements of different stakeholders including non-overlap of courses, free hours, lecturers' preferences, student preferences, etc. Furthermore, the course timetable can have a huge impact on queues in stair halls and elevators, particularly for universities or colleges with many students that follow courses in a single building. The congestion problems in stair halls and elevators are caused by traveling students that all have to switch rooms at the same time between two consecutive lectures. Clearly, student flows can be controlled and monitored via the course timetables. For example, if the schedules are arranged so that consecutive lessons take place in rooms situated on the same floor (or on a floor as close as possible), there will be far fewer queues at the elevators and in the stairwells. Thus, next to the various constraints and preferences of different stakeholders, the resulting student flows should also be taken into account when building the course timetable.

Despite the large complexity in building course timetables, in many educational institutes course timetables are still developed manually, which requires a lot of time and creativity of the planners. It is nearly impossible for human planners to solve the enormous puzzle taking into account the constraints and preferences of all stakeholders, let alone to incorporate the resulting student flows. This paper therefore presents an automated approach to develop course timetables based on a two-stage integer programming (IP) approach. In the first stage, lectures are assigned to timeslots taking into account the various constraints and maximizing the

Jeroen Beliën
Center for Informatics, Modeling and Simulation, Hogeschool-Universiteit Brussel,
Warmoesberg 26, B-1000 Brussel, Belgium
Center for Operations Management, KULeuven, Naamsestraat 69, B-3000 Leuven, Belgium
E-mail: jeroen.belien@hubrussel.be

Annelien Mercy
Nationale Delcrederedienst
E-mail: annelien_mercy@hotmail.com

stakeholders' preferences, while the second stage allocates each lecture-timeslot combination to a room with the objective of minimizing the resulting student flows.

This research was motivated by the course timetabling problem at the Faculty of Economics and Management at the Hogeschool-Universiteit Brussel (HUB). In recent years, the HUB has gone through a process of campus consolidation in which several buildings at different locations in Brussels have been sold and the lectures of all economic programs have been concentrated at a single location in the center of Brussels. As a result, over 8000 students daily follow classes in a single building, which inevitably causes major problems in terms of congestion at the elevators and the stairwells during lecture transitions.

Since many HUB students travel to Brussels by train, avoiding free hours in the course timetable is of key importance. Unfortunately, incorporating free hours through hard or soft constraints in the IP models turned out to be computationally intractable for the HUB problem size. Therefore, we introduce the concept of masks. A mask is defined for each student group and is a set of timeslots during which the lectures that are taken by the corresponding student group need to be scheduled. Each mask contains exactly as many timeslots as the weekly number of lectures that are taken by the corresponding student group. The masks determine how many lectures (not which ones) will be scheduled consecutively and how many lectures have free hours in between.

2 Literature review

The university course timetabling problem (UCTP) can be defined as the construction of a weekly timetable in which all operational rules and requirements of the academic institution are met and as many wishes as possible of the staff and students are satisfied [7]. Various solution techniques have been proposed for automating the development of course timetables [6]. Overviews are given in [4], [5], [13], [3], [11], [9] and [10].

In the past, due to computational difficulties the use of mathematical programming for solving UCTPs has been limited to small size instances. However, thanks to strong advances in computer software and hardware, and in IP formulations, mathematical programming approaches for timetabling problems have become more popular ([7], [15]). Examples of IP formulations for UCTPs can be found in [8], [7], and [14]. One advantage of mathematical programming approaches is the ease of incorporating additional soft constraints [5].

To the best of our knowledge, the studies in [1], [2] and [12] are the only ones that, to a limited extent, incorporate student flows. In [1] and [2] students and faculty members are adequately spread over all the available timeslots by constraints that impose an upper bound on the number of students that follow classes (take exams) during each timeslot in order to reduce parking problems and traffic congestions. [12] formulates a soft constraint specifying that students must follow consecutive lectures as much as possible in the same room in order to avoid traveling students. Hence, they do not model student flows that depend on the distance (e.g., in terms of number of floors) between the rooms of consecutive lectures.

3 Two-stage integer programming approach

Our two-stage approach aims at building the course timetable for one semester. Both stages involve the solution of an IP model. The first stage assigns lectures to timeslots and rooms taking into account lecture preferences, room (capacity) restrictions, free hours, work regulations, event-clash constraints, etc. The goal of this first stage IP model is to minimize weighted violations of lecturers being scheduled during timeslots for which they prefer not to be scheduled. The second stage changes the room assignment of the lecture-timeslots from the first stage with the objective of minimizing the resulting student flows. In the second stage the lecture-timeslot combinations that result from stage 1 are assigned to a new room with the aim of minimizing the resulting student flows. The stage 2 objective function minimizes the number of students that have to use the elevator (or stairways) for at least two floors between

two consecutive lectures. We realize that this definition of student flows is somewhat arbitrarily. Therefore, alternative definitions of student flows have been studied as well.

4 Results

The course scheduling problem at the HUB involves 396 lectures, 436 student groups, 171 lecturers and 56 rooms (including 9 pc rooms). Only 31 lectures require a pc room. Both IP models were solved using IBM-ILOG CPLEX Optimization Studio 12.3 on a pc with a 2 GHz Intel® Core™ 2 Duo processor and 2 Gb RAM. The two-stage approach only requires six minutes of CPU time. To evaluate the quality of the solutions the results are compared with the manually developed (and used) course timetable for the second semester of the academic year 2011-2012. The results listed in Table 1 indicate that the automated two-stage approach outperforms the manual scheduling with respect to both the lecturers' preferences and the student flows.

Table 1: Comparison between the two-stage approach and the manual approach (based on 365 of the 396 lectures)

	Penalty cost lecturer preferences	Student flows
Two-stage approach	425	282
Manual	535	3,731

Table 2 compares the results of the manual and the two-stage model for alternative definitions of student flows. The first column of Table 2 repeats the results of the basic case. The second column shows that, when also incoming student flows at 08:30 and 13:30 are incorporated, the automated two-stage approach still succeeds in finding a course timetable with considerably reduced student flows compared to the manual approach. The results listed in the third column indicate that the automated two-stage approach even finds a solution of zero student flow when student flows between rooms that differ less than three floors are neglected.

Table 2: Comparison between the two-stage approach and the manual approach for alternative definitions of student flows (based on 365 of the 396 lectures)

	Student flows only between consecutive classes. Rooms are separated by at least two floors.	Student flows between consecutive classes and incoming flows at 08:30 and at 13:30. Rooms are separated by at least two floors.	Student flows only between consecutive classes. Rooms are separated by at least three floors.
Two-stage approach	282	4,181	0
Manual	3,731	6,870	1,991

References

1. Al-Yakoob, S. M., & Sherali, H. D. (2007). A mixed-integer programming approach to a class timetabling problem: A case study with gender policies and traffic considerations. European Journal of Operational Research, 180(3), 1028-1044.
2. Al-Yakoob, S. M., Sherali, H. D., & Al-Jazzaf, M. (2010). A mixed-integer mathematical modeling approach to exam timetabling. Computational Management Science, 7(1), 19-46.
3. Burke, E. K., Petrovic, S., Recent research directions in automated timetabling, European Journal of Operational Research, 140, 266-280 (2002)
4. Carter, M. W., Laporte, G., Recent developments in practical course timetabling. In: E. K. Burke & P. Ross (Eds.). Practice and theory of automated timetabling I, pp. 1-21. Springer, London (1996)
5. Carter, M. W., Laporte, G., Recent developments in practical course timetabling. In: E. K. Burke & M. W. Carter (Eds.). Practice and theory of automated timetabling II, pp. 3-19. Springer, London (1998)
6. Chiarandini, M., Socha, K., Biarattari, M., Rossi-Doria, O., An effective hybrid algorithm for university course timetabling, Journal of Scheduling, 9, 403-432 (2006)
7. Daskalaki, S., Birbas, T., Housos, E., An integer programming formulation for a case study in university timetabling, European Journal of Operational Research, 153, 117-135 (2004)
8. Dimopoulou, M., Miliotis, P., Implementation of a university course and examination timetabling system, European Journal of Operational Research, 130, 202-213 (2001)
9. Lewis, R., A survey of metaheuristic-based techniques for university timetabling problems, OR Spectrum, 30, 167-190 (2008)
10. MirHassani, S. A., Habibi, F., Solution approaches to the course timetabling problem, Artificial Intelligence Review, 1-17 (2011)
11. Petrovic, S., Burke, E. K., University timetabling. In J. Y-T. Leung (Ed.). Handbook of scheduling: Algorithms, models, and performance analysis, pp. 1-34. CRC Press, Boca Raton (2004)
12. Pongcharoen, P., Promtet, W., Yenradee, P., Hicks, C., Stochastic optimisation timetabling tool for university course scheduling, International Journal of Production Economics, 112, 903-918 (2008)
13. Schaerf, A., A survey of automated timetabling, Artificial Intelligence Review, 13, 87-127 (1999)
14. Schimmelpfeng, K., Helber, S., Application of a real-world university-course timetabling model solved by integer programming, OR Spectrum, 29, 783-803 (2007)
15. Wren, A., Scheduling, timetabling and rostering: A special relationship? In E. K. Burke & P. Ross (Eds). Practice and theory of automated timetabling V. Lecture Notes in Computer Science, pp. 46-75. Springer, Berlin (1996)

MISTA 2013

Solving the Hybrid Job Shop Problem with Sequence Dependent Setups

Stéphane Dauzère-Pérès · Liji Shen · Janis Sebastian Neufeld

1 Introduction

This study addresses the hybrid job shop scheduling problem where sequence dependent setup times are involved. Our objective is to find both assignments and sequences of operations which minimize makespan C_{max}.

In a classical job shop environment, each job i consists of a number of operations $(O_{ij}, j = 1, \ldots, n_i)$ which are assigned to one prescribed machine k. As an important extension, hybrid job shop systems allow operations to be carried out on a given set of machines $(\mathcal{M}_{ij})$ [2]. Due to the availability of alternative machines, hybrid job shop introduces additional flexibility into the standard job shop system. This is particularly crucial for improving scheduling performance and efficiency in modern production management and time-based competition.

Moreover, we consider sequence dependent setup times $(s_{ii'k})$, where the magnitude of setups depends on the current job i and its immediate predecessor i' on the same machine [1]. As one of the most difficult aspects in scheduling [3], sequence dependency is, nevertheless, highly relevant in practice. Applications are encountered frequently in manufacturing environments where a facility has to be shut down after completing a job, in order to be adjusted to a desired state for processing the next job. Typically, this changeover delay is determined by the similarity in technological requirements for the two consecutive jobs.

For this specific scheduling problem, we first present two mathematical formulations which are able to solve small problem instances to optimality. Since instances

Stéphane Dauzère-Pérès
École des Mines de Saint-Étienne 880, route de Mimet, 13541 Gardanne, France
E-mail: Dauzere-peres@emse.fr

Liji Shen
Department of Business Administration and Economics, Technische Universitaet Dresden, 01062 Dresden, Germany
E-mail: liji.shen@tu-dresden.de

Janis Sebastian Neufeld
Department of Business Administration and Economics, Technische Universitaet Dresden, 01062 Dresden, Germany
E-mail: janis_sebastian.neufeld@tu-dresden.de

of practical interest are computationally intractable, we also develop a sophisticated tabu search algorithm. In the coming sections, we briefly describe our formulations and algorithm.

2 Mathematical Formulations

First, we present a mathematical formulation which extends conventional models for shop floor scheduling problems. Decision variables are binary variables α_{ijk} indicating whether or not operation O_{ij} is assigned to machine k, and $\beta_{iji'j'}$ indicating operation sequences. In addition, t_{ijk} and p_{ijk} denote the start time and processing time of the corresponding operation, respectively.

$$\min C_{max}. \tag{1}$$

Subject to:

$$\sum_{k\in\mathcal{M}_{ij}} \alpha_{ijk} = 1 \quad \forall i = 1,\ldots,n, j = 1,\ldots,n_i, \tag{2}$$

$$t_{ijk} \geq s_{0ik} - (1-\alpha_{ijk})H \quad \forall i = 1,\ldots,n, j = 1,\ldots,n_i, k \in \mathcal{M}_{ij}, \tag{3}$$

$$t_{ijk} \geq t_{i(j-1)k'} + p_{i(j-1)k'} \quad \forall i = 1,\ldots,n, j = 2,\ldots,n_i, k \in \mathcal{M}_{ij}, k' \in \mathcal{M}_{i(j-1)}, \tag{4}$$

$$t_{ijk} \geq t_{i'j'k} + p_{i'j'k} + s_{i'ik} - (2-\alpha_{ijk}-\alpha_{i'j'k}+\beta_{iji'j'})H, \tag{5}$$

$$t_{i'j'k} \geq t_{ijk} + p_{ijk} + s_{ii'k} - (3-\alpha_{ijk}-\alpha_{i'j'k}-\beta_{iji'j'})H, \tag{6}$$

$$\forall i = 1,\ldots,n, j = 1,\ldots,n_i, i' = 1,\ldots,n, j' = 1,\ldots,n_{i'}, O_{ij} \neq O_{i'j'}, k \in \mathcal{M}_{ij} \cap \mathcal{M}_{i'j'}, \tag{7}$$

$$C_{max} \geq t_{in_ik} + p_{in_ik} \quad \forall i = 1,\ldots,n, k \in \mathcal{M}_{in_i}. \tag{8}$$

Next, we propose an alternative model based on disjunctive graphs. The binary variable $\gamma_{iji'j'k}$ equals 1 if there is an arc connecting the two operations O_{ij} and $O_{i'j'}$ on machine k, and 0 otherwise. Nodes 0 and $*$ are two fictitious nodes in the graph.

$$\min C_{max}. \tag{9}$$

Subject to:

$$\sum_{i=1}^{n}\sum_{j=1}^{n_i} \gamma_{0,ijk} \leq 1 \quad \forall k \in \mathcal{M}_{ij}, \tag{10}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n_i} \gamma_{ijk,*} \leq 1 \quad \forall k \in \mathcal{M}_{ij}, \tag{11}$$

$$\sum_{\substack{i=1\\ i\neq i'}}^{n}\sum_{j=1}^{n_i} \sum_{k\in\mathcal{M}_{ij}\cap\mathcal{M}_{i'j'}} \left(\gamma_{iji'j'k} + \gamma_{0,i'j'k}\right) = 1 \quad \forall i' = 1,\ldots,n, j' = 1,\ldots,n_{i'}, \tag{12}$$

$$\sum_{\substack{i=1\\ i\neq i'}}^{n}\sum_{j=1}^{n_i} \sum_{k\in\mathcal{M}_{ij}\cap\mathcal{M}_{i'j'}} \left(\gamma_{i'j'ijk} + \gamma_{i'j'k,*}\right) = 1 \quad \forall i' = 1,\ldots,n, j' = 1,\ldots,n_{i'}, \tag{13}$$

$$t_{ijk} \geq s_{0ik} - (1-\gamma_{0,ijk})H \quad \forall i = 1,\ldots,n, j = 1,\ldots,n_i, k \in \mathcal{M}_{ij}, \tag{14}$$

$$t_{ijk} \geq t_{i'j'k} + p_{i'j'k} + s_{i'ik} - (1-\gamma_{i'j'ijk})H,$$
$$\forall i = 1,..,n, j = 1,..,n_i, i' = 1,..,n, j' = 1,..,n_{i'}, O_{ij} \neq O_{i'j'}, k \in \mathcal{M}_{ij} \cap \mathcal{M}_{i'j'}, \tag{15}$$

$$t_{ijk} \geq t_{i(j-1)k'} + p_{i(j-1)k'} \quad \forall i = 1,\ldots,n, j = 2,\ldots,n_i, k \in \mathcal{M}_{ij}, k' \in \mathcal{M}_{i(j-1)}, \tag{16}$$

$$C_{max} \geq t_{in_ik} + p_{in_ik} \quad \forall i = 1,\ldots,n, k \in \mathcal{M}_{in_i}. \tag{17}$$

3 Tabu Search Algorithm

We next present a tabu search algorithm, which incorporates various neighbourhood functions, triple tabu lists and multi-level diversifications. With regard to neighbourhoods, they are constructed after fully examining structural as well as dominance properties. Corresponding tabu definitions and tabu lists are then developed to avoid being trapped in cycles. Various methods for achieving diversity, on the other hand, compensate the intensified neighbourhood search.

Of particular note is that the algorithm contains feasibility check and lower bound calculation, which are based on several theorems and lemmta. As a result, the search procedure can directly exclude futile moves. Promising solution spaces can thus be quickly located.

4 Computational Results

The two mathematical formulations proposed in section 2 are implemented in LINGO and CPLEX. Small instances with up to 10 jobs on 5 machines are solvable by using reasonable amount of computing time.

The tabu search algorithm is implemented in C++ programming language. Preliminary computational experiments show that our algorithm finds competitive solutions for benchmark instances without setups. Test results also confirm the effectiveness of integrated properties. First, they succeed in integrating the assignment decision into the scheduling problem. By identifying infeasibilities and calculating lower bounds in advance, computing time is remarkably reduced.

References

1. T.C.E. Cheng and Gupta, J.N.D. and G. Wang, A review of flowshop scheduling research with setup times, *Production and Operations Management*, **9**, 262–282 (2000)
2. S. Dauzère-Pérès and J. Paulli, An integrated approach for modelling and solving the general multiprocessor job-shop scheduling problem using tabu search, *Annals of Operations Research*, **70**, 281–306 (1997)
3. M. Laguna, A heuristic for production scheduling and inventory control in the presence of sequence-dependent setup times, *IIE Transactions*, **31**, 125–134 (1999).

MISTA 2013

Roster Evaluation Based on the Classifiers for the Nurse Rostering Problem

Vaclavik Roman · Sucha Premysl · Hanzalek Zdenek

1 Introduction

The Nurse Rostering Problem (NRP), which is a specific task of the staff assignment problem, is a well known NP-hard combinatorial problem. It can be solved by many different algorithms, both exact or heuristic [1,5]. Good rosters have a significant impact on the quality of health care, the satisfaction of nurses, the hospital budget, etc.

Hospitals need to create these rosters periodically (e.g. monthly) while all conditions remain more or less the same. Usually a heuristic rostering algorithm has to evaluate many intermediate and usually useless solutions, which is a very time consuming task (about 80% of the total algorithm runtime - based on our experiments). Furthermore, almost all immediate solutions are dropped and never used again, not even in the current run of the scheduling algorithm. This implies that the scheduling algorithms are generally not able to learn anything from the problem instance structure.

The paper is focused on using classifiers which are learned from the historical data of the problem instance and are able to quickly recognize a good/bad roster of the same or the similar instance. Thus, the contribution of this work is a design of the pattern based classifiers/combination of classifiers which significantly reduces the use of a time consuming evaluation of a cost-oriented objective function while the solution quality remains comparable.

2 Related works

The first applications of learning techniques were appeared in the articles in the 90's. W. Zhang et al. [11] described a search policy for the resource-constrained scheduling problems. A neural network was used as an approximator for a future value of the resource dilation factor. Li et al. [6] used a neural network and logistic regression to determine the right order of execution of low-level heuristics in a hyper-heuristic method on the exam timetabling problems. Both approaches were able to speed up

R. Vaclavik · P. Sucha · Z. Hanzalek
Department of Control Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Prague, Czech Republic
E-mail: vaclarom@fel.cvut.cz, suchap@fel.cvut.cz, hanzalek@fel.cvut.cz

search of the algorithm. Li et al. [7] also introduced a neural network as a tool to distinguish between good and bad solutions. Unfortunately, the neural network is based on the structure of the whole roster which means that a different classifier must be learned if a change is made in the problem instance (e.g. the number of employees can vary for each scheduling period). Moreover, the authors use the best known objective function values from the literature to determine whether a roster is good or bad which is unrealistic since this knowledge cannot be gained *a priori*. In this work we suggest improvements eliminating these problems.

3 Problem statement

The goal of the Nurse Rostering problem is to assign shifts to nurses for each day in the scheduling period. Each nurse cannot work more than one shift each day [3,8].

The problem is parameterized by the number of nurses n, the number of days during the planning period d and the number of shifts s. Then the roster R is a binary matrix such that $\forall i \in \{1 \dots n\}, \forall j \in \{1 \dots d\}, \forall k \in \{1 \dots s\}$ $R_{ijk} = 1$ iff shift k is assigned to nurse i on day j and 0 otherwise. The quality of roster R is given by the objective function Z which is defined as $Z(R) = (\sum_{i=1}^{n} Z_i(R)) + C(R)$, where Z_i is the quality of the assignment related to nurse i and C is the cover penalty (=0 if all shifts are assigned).

During creating (optimization) of the roster, two types of constraints are considered. Hard constraints have to be fulfilled always. On the other hand, soft constraints may be violated but these violations are reflected in the objective function Z.

In this paper, we illustrate our approach on a general Tabu Search algorithm (TS) which keeps $C(R)$ the same. The problem of TS, and also other methods, is a cost-oriented objective function evaluation. The algorithm typically spends 80% of its whole runtime on evaluation of Z. In fact, the majority of these solutions are useless and therefore we propose a mechanism, based on pattern recognition, enabling the disposing of them.

4 Pattern recognition for the roster evaluation

The pattern recognition techniques, in our case classifiers, are able to quickly decide whether the structure of the solution is bad or good [4]. The *classifier* defines a set of *classes* characterizing the input data. Its aim is to assign the correct class to all input vectors. This assignment is based on the experience gained during the *learning phase* on the *training data*. We chose a neural network as a tool for the classifier realization [2,10].

In our approach, the input vector of the classifier is regarded as a change in the roster of one employee i. More specifically, it is the roster of employee i before the change and the same roster after the change. We consider several classifiers differing in the shift assignment representation. The first one (classifier1) represents a shift as a real number, e.g. an early shift is represented as 0.3, a late shift as 0.6, etc. The second one (classifier2) uses a binary encoding defined by R_{ijk}, i.e. one input of the classifier is one R_{ijk} for given employee i. There are other possibilities, but according to our experiments these two were the most successful in terms of the *classification rate*. The classification rate is the ratio of the number of correctly classified samples to the total

number of samples. The output of the classifiers corresponds to the Z improvement or deterioration. Since TS usually exchanges shifts between two employees, the classifier is ran for both of them and the overall improvement is estimated using these two results.

Boosting algorithms, namely Adaptive Boosting (AdaBoost), can be used to achieve a better classification rate. In general, it combines weak classifiers (e.g. classifier1, classifier2) to create one strong classifier of which output is the weighted sum of the outputs of the weak classifiers. The weights of the weak classifiers are adjusted during the learning phase of AdaBoost [9].

Since any classifier or any boosting algorithm will never have a 100% classification rate, the classifier/AdaBoost is used in the Tabu Search as a filter which eliminates the majority of potentially bad solutions while the rest of these solutions are evaluated by the standard objective function Z.

5 Experimental results

We verified our approach on the benchmark instances from http://www.cs.nott.ac.uk/~tec/NRP/, namely "Valouxis", "Millar", "Millar-s", "Ortec" and "GPost". 2-layer neural networks were used for all classifiers. Each neuron used a sigmoid activation function and the hidden layer consisted of 10 neurons. Data for the neural network learning was obtained from 500 Tabu Search runs using different initial rosters for each benchmark instance. From that set of data 10000 unique samples were selected with uniform distribution to the classes (70% training data, 30% testing data). The neural network was learned by the backpropagation method with learning rate 0.3 and 100 iterations (epochs). All classifiers including AdaBoost were used as a filter which eliminates 90% of potentially bad solutions in each iteration of Tabu Search. AdaBoost was learned in 2 iterations, i.e. the combination of 2 classifiers. The learning phase of classifier1, classifier2 and AdaBoost took approximately 2, 8 and 2 seconds under the circumstances. Experiments with the TS were performed on initial solutions which were different from those used in the learning phase.

Table 1 Experiments with the trained classifiers

Instance	CPU time [s]				Deviation in solution quality		
	class.1	class.2	AdaBoost	standard	class.1	class.2	AdaBoost
Valouxis	0.39	0.88	1.29	3.86	-1	-1	-1
Millar	0.03	0.04	0.06	0.11	0	0	-1
Millar-s	0.02	0.02	0.04	0.07	0	0	-1
Ortec	1.89	2.96	3.89	5.23	+2	+2	+1
Gpost	0.38	0.51	0.97	1.07	+3	+2	+1
Average	0.54	0.88	1.25	2.07	+0.8	+0.6	-0.2

Table 1 shows the comparison between the cost-oriented objective function evaluation, classifier1, classifier2 and AdaBoost. In the first column, there is the name of the benchmark instance. The following 4 columns show the CPU time consumed by the given evaluation methods. The last 3 columns represent the deviation in the solution quality (i.e. the difference in the number of soft constraint violations) while the roster from the standard cost-oriented evaluation was considered as the reference solution.

Overall, the experiments reveal a good classification rate: 82.65% for classifier1, 86.92% for classifier2 and 89.85% for AdaBoost. The average speedup for the single classifiers is roughly 3.5 and for AdaBoost 2.

6 Conclusion

The paper shows the usage of the neural network as a classifier on the Nurse Rostering Problem. This approach is employed as a filter which eliminates the most potentially bad solutions. The remaining solutions are then evaluated by the standard cost-oriented objective function. The trained classifier/AdaBoost can be successfully used on the same instances with different data from the training data and also on the similar data with added/removed soft constraint/employee(s). AdaBoost is slower than the single classifiers but on the other hand it is more accurate, i.e. a compromise between the better roster quality and a longer processing time of the scheduling algorithm.

7 Acknowledgments

This work was supported by the ARTEMIS initiative funded by the European Commission and the Ministry of Education of the Czech Republic under the project DEMANES 295372.

References

1. U. Aickelin and K. A. Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *CoRR*, abs/0802.2001, 2008.
2. C.M. Bishop and G. Hinton. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.
3. E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *J. of Scheduling*, 7(6):441–499, November 2004.
4. C.H. Chen. *Handbook of Pattern Recognition and Computer Vision: 4th Edition*. Imperial College Press, 2010.
5. R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
6. J. Li, E. K. Burke, and R. Qu. Integrating neural networks and logistic regression to underpin hyper-heuristic search. *Know.-Based Syst.*, 24(2):322–330, March 2011.
7. J. Li, E. K. Burke, and R. Qu. A pattern recognition based intelligent search method and two assignment problem case studies. *Applied Intelligence*, 36(2):442–453, March 2012.
8. S. Petrovic and G. Vanden Berghe. Comparison of algorithms for nurse rostering problems. In *Procedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*, pages 1–18, 2008.
9. G. Ratsch, T. Onoda, and K.-R. Muller. Soft margins for adaboost. *Machine Learning*, 42:287–320, 2001.
10. B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 2008.
11. W. Zhang and T. G. Dietterich. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Technical Report, Department of Computer Science, Oregon State University.*, 1997.

MISTA 2013

On the effect of complexity and uncertainty in ELSP

Raúl Cortés • Pablo Molina• José Pedro Garcia-Sabater

1 Introduction

Studies into the Economic Lot Scheduling Problem (ELSP) have typically focused on generating mathematical models to describe it for one thing, and to define and analyze the behavior of the resolution procedures for another. In this paper we analyze whether there is a relationship between total costs obtained by solving the theoretical model and those obtained by simulation. Our hypothesis is that the relationship, if any, depends on the degree of complexity and uncertainty in the initial data set in each case. Once this relationship is defined, it can be used to introduce new elements of analysis in designing control algorithms suitable for each particular version of the ELSP. It is worth stressing that we employ the term complexity in this work in relation to the difficulty of reaching and implementing a feasible solution for a given instance of the framework problem and in terms of the initial data, and not in relation to the algorithmic complexity of the mathematical problem posed (which actually has been proven to be NP-Hard; [1]. Hence we are talking about the static or the structural complexity of the problem [2].

Evidently, the number of items is a driver of complexity, which has a direct effect on the costs for each instance in the same framework problem. The homogeneity of the products, in relation to some of their characteristics (demand levels, setup times, production ratios, etc.), similarly affects the problem's complexity. Moreover, as Bomberger 1966 [3] analyzed, the level of demand, and consequently the utilization factor, directly affects the problem's complexity as costs go up. Regarding uncertainty, it is frequent to find, in real situations in industry where ELSP arises, uncertainty in relation to the behavior of some data throughout the planning horizon. Indeed, a distinction is made in the literature between the classic ELSP problem (deterministic) and the Stochastic ELSP problem (SELSP) (for surveys, see [4] and [5]). Normally, behavior of demand, and that of production times and setup times, is considered subject to stochastic processes. As a preliminary analysis, this work considers both number of items and level of demand to be drivers of complexity, while it contemplates the effect of uncertainty on demand to be a driver of uncertainty.

2 Problem description

The classic ELSP problem consists in programming several items in a single facility in which only one product can be produced at the same time. Along the present work the following notation will be used:

i,n	Item indexes i,n=1..g
d_i	Demand rate
p_i	Production rate

Raúl Cortés Fibla
Departamento de Organización de empresas, Univesitat Politécnica de Valencia, Cno. De Vera s/n, Valencia, spain
E-mail: racorfi@omp.upv.es
Pablo Molina Morte
Grupo de Investigación ROGLE, Univesitat Politécnica de Valencia, Cno. De Vera s/n, Valencia, spain
E-mail: pablomolinamorte@gmail.com
Jose Pedro García Sabater
Grupo de Investigación ROGLE, Univesitat Politécnica de Valencia, Cno. De Vera s/n, Valencia, spain
E-mail: jpgarcia@omp.upv.es

A_i Setup Cost
h_i Holding Cost
c_i Setup time
T_i Cycle length
I_i Available inventory of item i
TC Total Cost

Since the original definition of the problem, different approaches have been made to solve the ELSP regarding the cycle length definition. This work employs the Independent Solution (IS) [6] and the Common Cycle Approach (CC) [7] methods.

In order to solve the problem of defining a production schedule in ELSP, many heuristics in the literature have been defined [8],[9],[10]. In any case, the most commonly applied rule is that set out by Segerstedt, 1999 [11], based on the cover time, defined as:

$$RO_i = \frac{I_i}{d_i} \quad (1)$$

This work involves the classic ESLP problem and employs the IS and CC techniques as methods to determine the cycle length, as well as a ROI-based heuristics to establish the sequence (see Section 3). In the simulation study, the stochastic behavior of demand is introduced by considering that weekly demand is given according to the values of normal distribution, centered on a weekly average value and a certain variation coefficient.

3 Methods

Different instances for the same problem have been defined, generating several data sets for the problem posed. In each instance, the costs corresponding to the theoretical model are evaluated, as are the costs obtained after the simulation process. In each experiment, the simulation of 224 problem instances is done. Sets of problems with *g=10, 20, 30, 40* items, and 4 levels of demand with the corresponding utilization factor: $\sum \frac{d_i}{p_i}$ *=0.22;0.44;0.66;0.88*, are defined. In each case, coefficients of variation of demand are used, ranging from 0 to 0.3, and in intervals of 0.05. For the set of problems generated, a similar structure to Bomberger's problem for 10 items is deliberately used. The same demand and production ratio, the same setup times and the same storage and setup costs per item are employed. In order to introduce the "no. of items" driver of complexity, the number of items has been multiplied by *n=1, 2, 3* and *4* for the problem with *g=10* items, lowering the demand of each one according to the same factor. To evaluate the solution of various scenarios, a simulation model with discrete events is developed with the SIMIO software.

Sequencing the works in the simulation is run according to two procedures: *Fixed + Dynamic cycle length*, and *dynamic*. With the former procedure, works are launched only when it is their turn in the sequence and their stock is under the reorder stock level. With the *Dynamic* procedure; a work is launched, every time the stock is under the reorder stock level. Should two itmes be in this situation during the same period, the order is launched according to the lower cover time rule, described in Section 2. Should no item fulfill these conditions, no order is launched.

The technique applied to evaluate holding costs after the simulation process, as opposed to those relating to the theoretical problem, is fundamental to understand the results of the present work. Evidently, the simulation process provides different costs to the theoretical ones, depending on the parameters in the problem, which are not considered in the model; e.g.: initial stocks; sequencing and launching rules; non delivery costs. This work uses a technique which eliminates the effect of non delivery costs by recalculating the level of stock required per article ("base stock") once simulation is complete, to avoid stock outs.

Table 1 shows the sets of experiments run under the previously described conditions in terms of the different criteria set for initial stock, reorder stock level and production sequence.

Experiment	Initial Stock	Reorder Stock Level	Production sequence
1	$\sum_{n=1}^{i}\left(c_n+\frac{d_n.T_n}{p_n}\right)\cdot d_n$	$(c_i+\frac{d_i.T_i}{p_i})\cdot d_i$	*Dynamic*
2	$\sum_{n=1}^{i}\left(c_n+\frac{d_n.T_n}{p_n}\right)\cdot d_n$	$(c_i+\frac{d_i.T_i}{p_i})\cdot d_i$	*Fixed + Dynamic cycle length*
3	$\sum_{n=1}^{i}\left(c_n+\frac{d_n.T_n}{p_n}\right)\cdot d_n$	$\left(c_i+\frac{d_i.T_i}{p_i}\right)\cdot d_i \quad \nabla i=1;\ d_i\cdot T_i \quad \nabla i>1$	*Fixed + Dynamic cycle length*

Table 1. Set of experiments

4 Results

Simulation was performed to evaluate the effect on the costs of the various drivers introduced.

Increased demand directly affects costs, given the expressions (2) (for IS) and (3) (for CC), see Vidal-Carreras [12]:

$$TC=\sum_{i=1}^{g}\sqrt{2A_ih_id_i\left(1-\frac{d_i}{p_i}\right)} \quad (2)$$

$$TC=\sqrt{\sum_{i=1}^{g}2A_ih_id_i\left(1-\frac{d_i}{p_i}\right)} \quad (3)$$

For the theoretical model, and for the specific case of 10 items, figure 1 (a) represents this behavior. Nonetheless, when running simulation under the conditions described for Experiment 1, considering the case of 10 items and not considering uncertainty in demand (coeff. of variation =0), a sharp growth of costs appears, specially for the IS approach, even with relatively low utilization factor. The same effect continues to appear, even when using a sequencing rule which is adapted to the procedure to calculate the cycle time (as is experiment 1 for CC approach). In this case, it is possible to confirm that the relative increase in the simulation costs is much lower (21% for utilization factor of 88.24).

The effect of the number of items on management costs is also given by the expression (2) and (3). After running simulation under the conditions of Experiment 1, the results provided in Figure 1 (b) are obtained (for the sake of clarity, only data with $\sum\frac{d_i}{p_i}=44.12$ are shown). In this case, the behavior of costs after simulation follows a similar rule when the number of items increases, which does not change when faced with increases in utilization factor. A gap appears only in holding costs, with a very slight percentage growth at increased utilization.

Increased demand implies increased total costs in relation to $\sqrt{d_i}$., as seen when considering the effect of level of demand in Expressions (2) and (3). However, if the behavior of the costs per demand unit is considered, the inverse behavior is seen (according to $\frac{\sqrt{d_i}}{d_i}$), as shown in figure 1 (c). In theory, this behavior can be extended to reach the system's maximum utilization, which is limited by the feasibility condition. Nonetheless when running the simulation under the conditions of experiment 1, the system accomplishes its saturation considerably earlier than expected, as evidenced by the appearance of a point of inflection on the curve for the IS approach, and a significantly lower reduction in unit costs for the CC approach, since the sequencing technique is more suitable, as described along this section. Actually the appearance of this change of pattern exactly indicates the level of utilization for which the system reaches its saturation under the current launching conditions. Again, this effect is less marked when more suitable sequencing techniques are used.

To check the behavior of the costs when faced with an increase in the uncertainty in demand, simulations are run under the conditions described for Experiments 1, 2 and 3. In Experiments 2 and 3, the sequence remains fixed, although the cycle time is dynamic according to the time that each item reaches the reorder stock (Fixed + Dynamic cycle length). Under these conditions, it is possible to check that management costs grow in the face of

greater uncertainty in demand, as Figure 1 (d) illustrates. Specifically for the simulation of the described experiments, it is not clear that under conditions with considerable uncertainty in demand, a fixed production schedule may present worse results regarding costs.

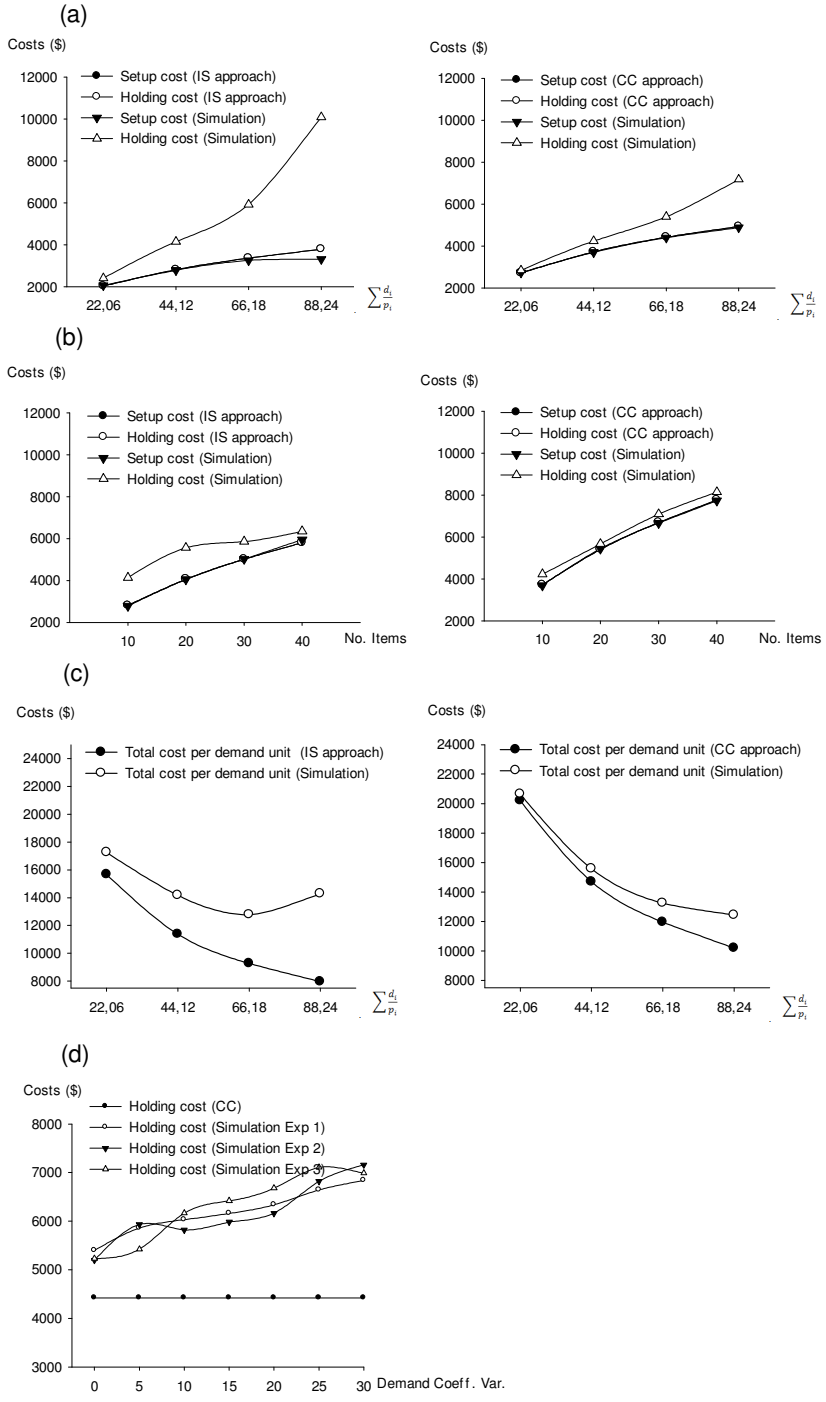


Fig 1. Cost behavior against increased demand (Model)

5 Conclusions

This work presents a methodology to analyze the behavior of total costs when simulating different instances in the ELSP problem when faced with variations in the various drivers of complexity and uncertainty affecting the problem. This procedure helps to quickly and intuitively determine the behavior of a simulation experiment by making the costs resulting from assuming real working conditions during the simulation have an impact on the storage costs by means of a "base stock" .By means of this technique, the results obtained for the behavior of the problem when faced with variations in utilization factor , no. of items and uncertainty in demand, are described. Moreover, the analysis of the behavior of the costs per demand unit is introduced as a tool to identify the system's saturation level when faced with increased demand. This analysis enables the identification of the level of limited usage for certain working conditions (sequencing heuristics, reorder stocks, etc.) in the face of increased demand. In line with this work, in the future, we intend to conduct an in-depth study into using the described methodology to analyze other drivers of complexity and uncertainty in the face of distinct variants of the problem and different solution procedures.

Acknowledgements The work described in this paper has been supported by the project ''CORSARI MAGIC DPI2010-18243'' of the Ministerio de Ciencia e Innvovación of Government of Spain within the Program "Proyectos de investigación fundamental no orientada"

References

1. Hsu, W. L. On the General Feasibility Test of Scheduling Lot Sizes for Several Products on One Machine, Management Science, Vol. 29, pp. 93-105 (1983).
2. Frizelle, G.D.M.. Getting the measure of complexity. Manufacturing Engineer, 75, (6) 268-270 (1996)
3. Bomberger, E. E. A dynamic programming approach to a lot size scheduling problem. Management Science, Vol. 12, nº. 11, p. 778.(1966)
4. Sox, C. R.; Jackson, P. L.; Bowman, A.; Muckstadt, J. A. A review of the stochastic lot scheduling problem. International Journal of Production Economics,Vol. 62, nº. 3, pp. 181-200. (1999).
5. Winands E.M.M., Adan I.J.B.F., Van Houtum G.J., The stochastic economic lot scheduling problem: A survey, European Journal of Operational Research, Vol. 210, pp 1-9, (2011)
6. Harris, F. W. How many parts to make an once. Factory, The Magazine of Management, Vol. 10, nº. 2, pp. 135-6-152. (1913).
7. Hanssmann, F. Operations-Research in Production and Inventory Control. J. Wiley. (1962).
8. Delporte, C. M.; Thomas, L. J. Lot Sizing and Sequencing for N-Products on One Facility. Management Science, Vol. 23, nº. 10, pp. 1070-1079. (1977).
9. Gallego, G.; Moon, I. The Effect of Externalizing Setups in the Economic Lot Scheduling Problem. Operations Research, Vol. 40, nº. 3, pp. 614-619. (1992).
10. Zipkin, P. H. Computing Optimal Lot Sizes in the Economic Lot Scheduling Problem. Operations Research, Vol. 39, nº. 1, pp. 56-63. (1991).
11. Segerstedt, A. Lot sizes in a capacity constrained facility with available initial inventories. International Journal of Production Economics, Vol. 59, nº. 1-3, pp. 469-475. (1999).
12. Vidal-Carreras, Pilar I. Enfoques para la resolución del problema ELSP. Working Papers on Operations Management Vol 1, Nº 2 (31-43) (2010)

MISTA 2013

Guiding wiki crowds through resource scheduling

Ioanna Lykourentzou • Wassila Aggoune-Mtalaa • Dimitrios J. Vergados • Yannick Naudet

1 Motivation

Wikis are among the most popular technologies for collaborative knowledge production. In brief, a wiki is a collaborative content creation system, where users contribute knowledge content in the form of articles, while they can also edit and even delete the contributions of others [1]. Wikis have received significant interest in the past few years and they are increasingly being used to support knowledge development in many domains, from education, to scientific research, and from activities of the public sector to enterprise environments. Last, one of the most well-known and studied wikis, the popular Wikipedia, is an ever-growing source of information with millions active users and articles.

The rapid expansion and success of wikis is based on the open form of user collaboration that they are based on. That is, wiki users are free to edit any article they wish, with almost no restrictions on their access and edit rights. This open collaboration enables the massive production of wiki articles, which cover a broad spectrum of topics and expertise backgrounds. However, this same self-coordination poses significant limitations in terms of content quality. Take as an example Wikipedia: although it features a number of qualitative articles, it has also a very long tail of low-quality ones [2]. This inability to guarantee quality lowers the reliability of wikis and hinders their adoption.

To guide the wiki crowd, systematize contributions and help them utilize their knowledge competencies more efficiently [3], we propose a coordination scheme that can be viewed as a scheduling problem [4]. The wiki is seen as a system with resources, which are the users and their expertise, and tasks, which are the wiki articles that need quality improvement. The objective is to match users to articles, in such a way as to maximize the average quality of the articles inside the wiki, regarding specific constraints such as user workload.

The work is presented as follows: Section 2 formulates the problem and gives the methodology adopted to solve it. Section 3 shows some promising results. Last, Section 3 concludes the work and gives perspectives for the future.

Ioanna Lykourentzou, Wassila Aggoune-Mtalaa and, Yannick Naudet
Public Research Centre Henri Tudor, Luxembourg
E-mail: {ioanna.lykourentzou, wassila.mtalaa, yannick.naudet}@tudor.lu
Dimitrios J. Vergados
NTNU, Norway
E-mail: dimitrios.vergados@item.ntnu.no

2 Problem formulation and Methodology

The wiki scheduling problem can be formulated as follows. Given a set of:

- wiki articles $A = \{i_1, i_2, \dots, i_{|A|}\}$, with quality below the threshold. Each article i has two characteristics: a current article quality q_i which changes after a new user contribution and a knowledge topic $D_i \in D = \{D_1, D_2, \dots, D_{|D|}\}$. Each article is considered to belong to exactly one knowledge topic.
- wiki users $U = \{u_1, u_2, \dots, u_{|U|}\}$. Each user u_j has an *expertise vector* $\boldsymbol{e_j}$, with length equal to the knowledge domains $|D|$. Expertise indicates the quality that the system estimates that user u_j can bring to an article belonging to a specific domain. The user expertise can be computed using past user data (e.g. ratings of past user contributions, with the help of Feed Forward Neural Networks, as detailed in [5]),

the problem is to find which article should each user be requested to contribute to, so that the average quality of articles inside the wiki C is maximized:

$$C = \frac{\sum_{i=1}^{|A|} q_i}{|A|} \quad (1)$$

The constraints considered in this problem formulation are:

- all articles need to surpass a given quality threshold T, $0 < T < 10$. This threshold is considered to be fixed for all the domains, but in a more general problem setting it could be set individually for each domain, depending on the community requirements.
- a user can be recommended only one article at a time. The maximum number of articles that can be given to him is therefore a binary workload $w = \{0,1\}$.
- Non-preemptive process. Once a user has been assigned with an article, he cannot be interrupted, to be recommended with another one.

The wiki problem has an additional complexity compared to classical scheduling problems. First there is *uncertainty* regarding *resource availability*: users enter a wiki when they want, remain connected for as long as they like and they may or may not accept to make a contribution. Secondly there is *uncertainty* regarding *resource capacity* (user expertise): the mechanism cannot know a priori if an expert user will enter the system, or whether the users that will enter are not knowledgeable enough to improve a certain article. Thirdly, a wiki article might require the contributions of multiple users before reaching the quality threshold, which brings upon the *need for sequenced, chain scheduling*, with hands-off resource dependencies (the input of one user starts when another user has finished contributing to the same article). Due to the above constraints, the problem complexity is high. Therefore, in the scope of this paper it was decided to opt for a heuristic greedy algorithm: once a user enters the wiki, the mechanism should suggest him with an article that the system estimates, at that moment, that the user is best fit for. This solution is not expected to necessarily lead to optimality but, in case it can improve the current quality state of wiki systems, it can be judged as sufficient. The overall wiki coordination mechanism, featuring the scheduling algorithm, functions as follows. For every article that is inserted into the wiki, either as a new article or as a contribution to a previous article version, the mechanism evaluates the article's quality, as a single numerical value. The mechanism then compares this quality to a pre-defined quality threshold. If the article does not surpass it, then the article needs to be enhanced by an additional user. The selection of which user will be asked to contribute to which article is handled by the greedy algorithm. The process of successive article contributions, quality evaluations and user selections continues until the article surpasses the quality threshold.

3 Evaluation

To evaluate the proposed algorithm we model two systems:

- *Benchmark*. In this system, users enter the wiki, view and contribute to articles as they would in a typical wiki, without any recommendations. The system is calibrated to function similarly to the English Wikipedia[6],on various statistical parameters

including the arrival rate of users, the user expertise distribution, the user probability of viewing each specific article and the user probability of contributing to an article. This calibration was validated using two indicative factors: workload and quality.

- *Smart*. The smart system extends the benchmark system, by applying on it the greedy algorithm that suggests articles to users. Apart from that, the other characteristics are identical to those of the benchmark. We refer to the smart system as a CI (Collective Intelligence) system, in line with the definition given in [7], and to highlight the fact that the system improves the emergence of the collective intelligence of the involved community, by combining user and algorithmic intelligence.

Comparing the performance of the benchmark and the CI system (Fig. 2), we may observe that the scheduling algorithm results in a significant improvement in the objective function, i.e. on the average article quality. We may also observe that this improvement is quicker as compared with wiki articles which often reach adequate quality levels in a slow manner.

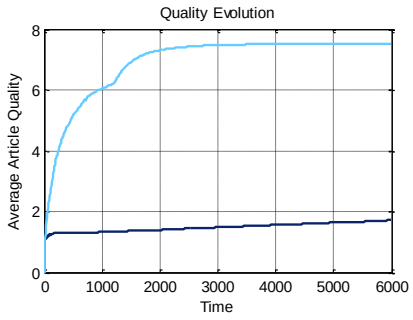


Fig.2. Evolution of the article quality achieved through the use of CI and benchmark systems

Another interesting feature to examine (Fig. 3a) is that the CI system leads to slightly more edits performed by participating users per article, compared to the benchmark system, however with a significant shift of the article quality distribution (Fig. 3b). Thus, the community manages to produce more qualitative articles through the use of the CI system, compared to the respective result achieved through the use of the benchmark system.

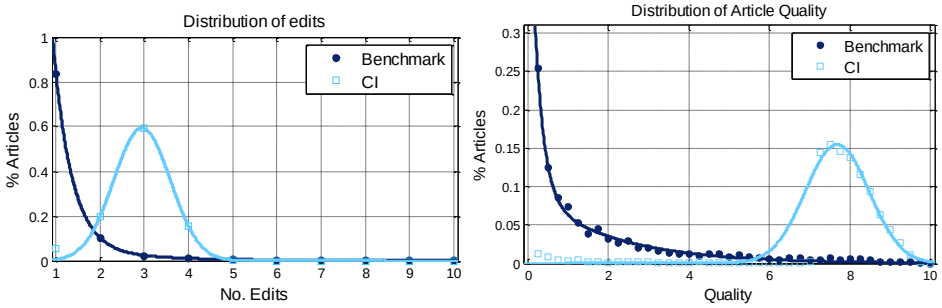


Fig.3. Comparison of distribution of a) edits and b) quality produced by the benchmark and CI system

The above results indicate that the scheduling-enabled version of the wiki system can help increase the produced article quality, better use expertise and reduce the time needed for the articles to reach satisfactory levels of quality.

4 Conclusions

In this work, we address the problem of low quality in wikis. A heuristic greedy algorithm is used to allocate users to the wiki articles that they can improve the most.

Experimental results show that this approach can indeed increase average article quality inside the wiki. Finally, viewing user coordination in modern web 2.0 systems as a resource scheduling problem can be extrapolated to other crowd-involving domains, such as crowdsourcing and this is a very interesting future research direction.

Acknowledgements: The work of I. Lykourentzou is supported by the National Research Fund, Luxembourg and co-funded under the Marie Curie Actions of the European Commission (FP7COFUND).The work of D. J. Vergados is supported by the ERCIM "A. Bensoussan" Fellowship.

References

1. P. Louridas, Using Wikis in Software Development, IEEE Software, 23, 88-91 (2006).
2. S.K. Lam and J. Riedl, Is Wikipedia growing a longer tail?, Proc. Proceedings of the ACM 2009 international conference on Supporting group work, ACM (2009).
3. A. Kittur, et al., Coordination in collective intelligence: the role of team structure and task interdependence, Proc. Proceedings of the 27th international conference on Human factors in computing systems, ACM (2009).
4. R. Lewis, A survey of metaheuristic-based techniques for University Timetabling problems, OR Spectrum, 30, 167-190 (2008).
5. I. Lykourentzou, et al., CorpWiki: A self-regulating wiki to promote corporate collective intelligence through expert peer matching, Inf. Sci., 180, 18-38 (2010).
6. F. Ortega, "Wikipedia: A quantitative analysis," Doctoral Thesis, Departamento de Sistemas Telemáticos y Computación (GSyC), Universidad Rey Juan Carlos, Móstoles, Spain, 2009.
7. T.W. Malone, et al., Harnessing Crowds: Mapping the Genome of Collective Intelligence, MIT Center for Collective Intelligence, 2009.

MISTA 2013

Batch Sizing and Just-In-Time Scheduling: Analysis of Some Special Cases

Öncü Hazır · Safia Kedad-Sidhoum

1 Introduction

Batch sizing deals with dividing the production into batches taking account the planned production quantities, which are derived from upper level lot sizing decisions. The literature mainly focused on splitting a batch of jobs into smaller lots so as to transfer and process different sub-lots at the same time on different machines. On the other hand, scheduling with both earliness and tardiness penalties cope with Just-In-Time (JIT) requirement of lower work in process. We address both these requirements.

We examine the production environments in which a product could not be released before its batch is totally completed [*batch availability*]. Potts and Wassenhove [6] described various single machine scheduling problems under batch availability and presented complexity results. Hall and Potts [3] addressed integrating scheduling and coordination in supply chains. Assuming that jobs are delivered to the customer in a single shipment, they optimize various delivery time based functions. Similarly, Selvarajah and Steiner [8] minimized the total inventory holding and delivery cost in a supply chain. Robert [7] and Chrétienne et al [2] developed dynamic programming algorithms to minimize total weighted tardiness penalties and setup costs. We refer to the papers of to Kreipl and Pinedo [5], Chen [1] for an overview of related problems and a survey of integrated supply chain scheduling studies.

Differently, Hazir and Kedad-Sidhoum [4] defined the "Batch Sizing and Just intime Scheduling (BSJS) Problem". This problem addresses JIT delivery while minimizing the setup costs induced by the deliveries as well as the JIT penalties. The batch size constraints have to be satisfied. Delivering on-time decreases the earliness and tardiness cost, while increases the setup cost because of smaller batches.

Let us consider K planned orders, which are ready at time zero. Each customer order k needs r_k units to be ready at a due date d_k. Assuming serial batching and batch

Öncü Hazır
TED Üniversitesi, İktisadi ve İdari Bilimler Fakültesi, Ziya Gökalp Caddesi No.48, 06420, Kolej, Çankaya, Ankara, Turkey.
E-mail: oncu.hazir@tedu.edu.tr

Safia Kedad-Sidhoum
LIP6, Université Pierre et Marie Curie. 4 place Jussieu, 75252 Paris, Cedex 05, France.
E-mail: safia.kedad-sidhoum@lip6.fr

availability, delivery time of the units of order k are dependent on the corresponding batch completion time, C_i for batch B_i. If $C_i < d_k$ (resp. $C_i > d_k$), a total earliness (resp. tardiness) penalty $\alpha_k(d_k - C_i)$ (resp. $\beta_k(C_i - d_k)$) is charged; where α_k (resp. β_k) represents the earliness (resp. tardiness) unitary penalty. To start a new batch, a setup cost π is assumed. Batch sizes are bounded with lower and upper limits (b,B).

An instance I of the single machine BSJS problem with K orders [K-BSJS] is denoted by $(\hat{r}, \hat{\alpha}, \hat{\beta}, \hat{d}, \pi, b, B)$ where $\hat{r} = (r_1, \cdots, r_K)$, $\hat{\alpha} = (\alpha_1, \cdots, \alpha_K)$, $\hat{\beta} = (\beta_1, \cdots, \beta_K)$ and $\hat{d} = (d_1, \cdots, d_K)$. On the other hand, a schedule Ψ represents a solution characterized byx $\Psi = (N, q, S)$, where N represents the total number of batches, q the batch production quantity matrix, q_i^k for $i = 1, ..., N$, $k = 1, ..., K$, and S is the vector of batch starting times $(S_1, \cdots, S_N)$. The variables and parameters N, r_k, d_k, b and B are discrete. Consequently, the problem reduces to find a feasible schedule that minimizes

$$f_I(\Psi) = \sum_{i=1}^{N} \sum_{k=1}^{K} q_i^k \; Max\left\{\alpha_k(d_k - C_i), \beta_k(C_i - d_k)\right\} + \pi N$$

We call a batch B_i *tardy* (resp. *early* or *on time*) if $C_i > d_k$ (resp. $C_i < d_k$ or $C_i = d_k$) for all orders k such that $q_i^k > 0$. B_i is a *straddling* batch if there exists an order k such that $q_i^k > 0$ and $S_i < d_k < C_i$. Let R_k be the cumulative requirement $\sum_{i=1}^{k} r_i$ of the first k orders, then the common due date d is *non-restrictive*, if $d \geq R_K = \sum_{i=1}^{k} r_i$, otherwise, it is *restrictive*. An instance of the problem is feasible if and only if $\lceil \frac{R_K}{B} \rceil \leq \lfloor \frac{R_K}{b} \rfloor$ [7].

2 Structural Properties

In this section, basic properties of the BSJS problem, which were previously proved (see [4]), will be summarized. Using them, we examine the structural characteristics of some special cases in Section 3, which is the contribution of this study.

Properties for the common due-date problem

*Property 1 (**No-Idle Time**)* A single block without idle times is dominant.

*Property 2 (**Straddling Batch**)* There exists an optimal schedule either starting at time zero or with a batch that completes at d. If d is non restrictive then no straddling batch exists.

Let us divide the batches of a schedule into two sets: the set $\mathcal{E}$ of early or on time batches and the set $\mathcal{T}$ of tardy batches. Therefore, any schedule contains $Q_{\mathcal{E}} = \sum_{i \in \mathcal{E}} q_i$ (resp. $Q_{\mathcal{T}} = \sum_{i \in \mathcal{T}} q_i$) early and on-time (resp. tardy) units.

*Property 3 (Λ **Shape Structure**)* The schedules in which the units in the batches $\mathcal{E}$ (resp. $\mathcal{T}$) are processed in non-decreasing (resp. non-increasing) order of α_k (resp. β_k) are dominant for instance I of K-BSJS.

*Property 4 (**SPT**)* There exists an optimal schedule for an instance I of K-BSJS in which batches in $\mathcal{E}$ (resp. in $\mathcal{T}$) are processed in non-decreasing order of their batch sizes (Shortest Processing Time).

Optimality of batches with almost equal sizes was proved [2] and a schedule was called *Almost Equal Batch Size (AEBS)* if the sizes of any two batches differ by at

most one. Similarly, we name a schedule *Almost At Limit Sizes (AALS)* if at most one batch has a size different than b and B. If the partial schedules $\mathcal{E}$ (resp. $\mathcal{T}$) is AALS (resp. AEBS), it is named $\mathcal{E}$-AALS (resp. $\mathcal{T}$-AEBS). AALS is based on the observation that large size early batches have less earliness costs.

*Property 5 (**AALS**)* A schedule is optimal for an instance I of K-BSJS if and only if it is $\mathcal{E}$-AALS.

Properties for the single order problem

*Property 6 (**Order Independence**)* Given the batch sizes in $\mathcal{T}$ (resp. in $\mathcal{E}$), the order of the batches in $\mathcal{T}$ (resp. in $\mathcal{E}$) does not affect the objective function.

*Property 7 (**AEBS**)* A schedule is optimal for an instance I of 1-BSJS if and only if it is $\mathcal{T}$-AEBS .

*Property 8 (**Convexity**)* When all the batches are tardy, the optimal total cost is a discrete convex function of n.

We denote Σ_0, the set of schedules with no straddling batch, Σ_1 and Σ_2, respectively, the set of schedules with a straddling batch such that $(r - S_{i_s}) \mod b = 0$ and $S_{i_s} \mod B = 0$ where S_{i_s} is the starting time of the straddling batch.

Property 9 $\Sigma_0 \cup \Sigma_1 \cup \Sigma_2$ is dominant for an instance I of 1-BSJS.

*Property 10 (**Starting Time**)* There exists an optimal schedule for an instance I of 1-BSJS such that either $S_1 = 0$ or $d - B - \lfloor \frac{\beta}{\alpha+\beta} r \rfloor \leq S_1 \leq d - \lceil \frac{\beta}{\alpha+\beta} r \rceil$.

3 Analysis of Special Cases for Common Due Date Problem

1. **Relaxing the upper bound on batch sizes**
   No upper bound is imposed on batch sizes; $B = \infty$. The problem is feasible as long as $R_K \geq b$. The optimal schedule has $|\mathcal{E}| = 1$, otherwise, combining the batches in $\mathcal{E}$ decreases earliness cost. In addition, using Property 2, we examine two cases for the optimal schedule: $S_1 > 0$ and $S_1 = 0$.
   (a) If $S_1 > 0$, then there is no straddling batch; there is a single on-time batch and no early batches. Therefore, orders will be processed in non-increasing β_k.
   All tardy batches have size b, otherwise, if there is a tardy batch that contains more than b items, a tardy unit can be moved to the on-time batch; starting this batch earlier is feasible ($S_1 > 0$) and less costly; which is contradictory.
   To find out the optimal batching, dynamic programming for $K > 1$ and binary search algorithm for case $K = 1$ could be used [2]. The complexity of binary search depends on the maximum number of batches, which is at most $\lfloor \frac{r}{b} \rfloor$.
   (b) If $S_1 = 0$ is optimal, $\mathcal{E}$ is composed of a single batch, either early or on-time, and $\mathcal{T}$ might contain a straddling batch. This makes the batching problem more difficult. However, for the case $K = 1$, we can use Property 9 to conclude that all tardy batches will have size b. The schedule contains a single early batch of size $d - b \left\lceil \frac{r-d}{b} \right\rceil$ and tardy batches with size b.

2. **Relaxing the lower bound on batch sizes**
   If $\pi = 0$, unit size batches are dominant for $\mathcal{T}$ and there is no straddling batch. All early batches will have size B and there will be an on time batch with size $1 \leq q_i \leq B$ (Property 5).
   Otherwise, dynamic programming for $K > 1$ and binary search algorithm for case $K = 1$ is needed to find out the sizes of batches [2].
3. **Relaxing the batch size constraints**
   As there is no constraint on splittability of batches, since $b = 1$ and $B = \infty$, the optimal schedule is such that $S_1 = 0$. There will be no early batches, hence no earliness penalty. As a result, the customer orders will be processed in non-increasing order of β_k .
   If $\pi \neq 0$, a straddling batch that starts at time zero is possible. However, if $\pi = 0$, there will be a single on time batch and succeeding tardy batches of unit size.
4. **No earliness penalties**
   In this case, only tardiness penalties are charged; customer orders are processed in non-increasing order of β_k. If $\pi = 0$, a maximum number of tardy batches will be created to minimize tardiness penalties.
5. **Fixed batch sizes**
   In this case, there is a fixed batch size equal to $b = B$. An instance of this problem first reduces to a feasibility problem. If the instance is feasible, for the single order problem, $K = 1$, the following property defines the schedule start time.

Corollary 1 *There exists an optimal schedule for an instance I of 1-BSJS such that either $S_1 = 0$ or $S_1 = d - \lceil \frac{\beta r}{B(\alpha+\beta)} \rceil B$.*

4 Special Cases for Distinct Due Dates

1. **No earliness and order independent tardiness penalty**
   Units are processed in Earliest Due Dates (EDD) order, i.e. $d_1 \leq d_2 \leq \cdots \leq d_K$.
   (a) $\pi = 0$, $b = 1$: Unit size batches clearly provide dominant schedules. An optimal schedule is thus obtained by processing (in the EDD order) the customer orders in unitary batches. The complexity of the corresponding algorithm is thus $O(K \log K)$, which is the cost of sorting the due dates.
   (b) $\pi = 0$, $B = b$: If $R_k = \sum_{i=1}^{k} r_i$ is splittable, an optimal schedule is obtained by processing (in EDD order) the customer orders in batches of size B, otherwise the instance has no solution. The complexity of the corresponding algorithm is again $O(K \log K)$.
2. **No earliness and order dependent tardiness penalties**
   In this case, EDD-schedules might not be dominant.
   (a) $\pi = 0, b = 1$: For this problem, there is an optimal schedule where all the batches are unitary. In addition, in the optimal schedule, $S_1 = 0$ and there is no idle time, hence the time horizon could be divided into R_K unitary intervals. The R_K batches, which we number as $i = 1, ..., R_K$, are assigned to the K customer orders (indexed by $k = 1, ..., K$). The assignment of batch B_i to customer order k creates the cost $c_{ki} = \max\{0, \beta_k(i - d_k)\}$ (Note that $C_i = i$). Consequently, the problem reduces to a minimum cost network flow problem with unitary arc capacities, in which orders are the supply nodes with capacity r_k and the orders are the demand nodes with unitary demands.

(b) $\pi = 0, B = b$: An instance of the problem is feasible as long as R_K is a multiple of B. If $R_K = N \times B$, we create N batches B_i, $i = 1, ..., N$. When a unit of order k is assigned to batch B_i, a cost $c_{ik} = \max\{0, \beta_k(iB - d_k)\}$ is charged; the problem becomes a transportation model so that the batches define the supply nodes (each with capacity B) and the orders represent the demand nodes with requirements r_k.

5 Conclusion

We summarized some theoretical results and dominance properties for the single machine BSJS problem and discussed some special cases. As an extension of this study, complexity of the open problems will be set, and efficient algorithms to solve difficult cases will be developed. Il will be also interesting to consider parallel machines environments.

References

1. Chen ZL (2010) Integrated production and outbound distribution scheduling: Review and extensions. Operations Research 58(1):130–148
2. Chrétienne P, Hazir O, Kedad-Sidhoum S (2011) Integrated batch sizing and scheduling on a single machine. Journal of Scheduling 14:541–555
3. Hall NG, Potts CN (2003) Supply chain scheduling: Batching and delivery. Operations Research 51(4):566–584
4. Hazir O, Kedad-Sidhoum S (2012) Batch sizing and just-in-time scheduling with common due date. Annals of Operations Research DOI 10.1007/s10479-012-1289-9
5. Kreipl S, Pinedo M (2004) Planning and scheduling in supply chains: An overview of issues in practice. Production and Operations Management 13(1):77–92
6. Potts CN, Wassenhove LNV (1992) Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity. Journal of Operational Research Society 43:395–406
7. Robert A (2007) Optimization des batches de production. PhD thesis, Université Pierre et Marie Curie (Paris 6)
8. Selvarajah E, Steiner G (2006) Batch scheduling in a two-level supply chain - a focus on the supplier. European Journal of Operational Research 173(1):226 – 240

MISTA 2013

Student Scheduling for Bachelor State Examinations

Beáta Kochaniková · Hana Rudová

1 Introduction

Bachelor state examinations take place at the Faculty of Informatics two times a year at the end of each semester. Each Bachelor student who wants to complete his/her Bachelor study must successfully complete the Bachelor state examinations. Since it necessitates the presence of many people at various times, timetabling must be accomplished. In the timetable, each student is assigned to a commission which typically consists of three people, the chair and two other examiners. Timetabling problem to be solved consists of two parts. First, commissions must be created and assigned to particular days of the examination week. Each commission is timetabled to one day and one room. There are four or five examination days and it is typically necessary to timetable from two to five commissions a day. The second part of the problem involves a specific student scheduling: assignment of students to committees in time such that it is acceptable for all involved teachers (examiners, supervisors and referees). Mostly, the exam of one student takes half an hour and contains two parts, thesis defense and exam based on the written preparation. Each student has its own thesis supervisor and referee of the thesis. Both these people must be able to attend the state exam of the student. An example of the timetable for one commission is demonstrated in Table 1. The timetable of each commission is given by a sequence of students split by a lunch break.

This problem was up to now solved manually with the help of the supportive graphical user interface [3]. One of the authors, Hana Rudová is responsible for state examination timetabling as a vice dean for Bachelor and Master studies. She has an experience with a manual construction of timetables for four semesters. In this paper we will describe results of the Bachelor thesis [2], which concentrates on the student scheduling part of the problem. The work describes main characteristics of the problem and discusses implementation of a new local search solver which has been applied

Beáta Kochaniková
Faculty of Informatics, Masaryk University, Brno, Czech Republic
E-mail: kochanikova@mail.muni.cz

Hana Rudová
Faculty of Informatics, Masaryk University, Brno, Czech Republic
E-mail: hanka@fi.muni.cz

Table 1 Example of the timetable for one commission.

Commission: doc. RNDr. Petr Hliněný, Ph.D. (chair)
Mgr. Aleš Křenek, Ph.D.
RNDr. Jaroslav Pelikán, Ph.D.
Thursday 21. 6. 2012, room B411

	Student	Supervisor	Referee
8:30 – 9:00	Rentka Michal	Škrabálek	Pavlovič
9:00 – 9:30	Štěrba Jakub	Osovský	Pavlovič
9:30 – 10:00	Kuročenko Andrej	Pavlovič	Kolář
10:00 – 10:30	Forman Jiří	Ráček	Kolář
10:30 – 11:00	Kolář Milan	Ludík	Ráček
11:00 – 11:30	Svitková Monika	Ráček	Ludík
11:30 – 12:00	Janošek Václav	Ludík	Ráček
12:00 – 13:00			
12:00 – 13:30	Laštůvka Zdeněk	Ludík	Ráček
13:30 – 14:00	Šimanský Dávid	Ráček	Toth
14:00 – 14:30	Jurč Michal	Ráček	Toth
14:30 – 15:00	Škrabal Jan	Pelikán	Dohnal
15:00 – 15:30	Coufal Jakub	Pelikán	Dohnal
15:30 – 16:00	Lenčo Milan	Křenek	Kulhánek

on two real-life data instances. Timetables generated by this prototype solution have been compared with manually created timetables and they have been shown to be comparable with them.

2 Characteristics of the Problem

We will describe the main constraints which are important for our problem. The base hard constraints are related with the structure of the schedules.

(C1) There is one commission in each room each day.
(C2) There is one student scheduled in each commission in one time period (with exception of two lunch break periods which are free).

The problem consists of hard constraints requesting the presence of each *teacher* (examiner, supervisor or referee) in one place only and the presence of each supervisor and referee in the exam of their students. Also it is not possible to schedule same teachers in the consequent time periods in different rooms since even a small latency would cause a teacher absence.

(C4) The supervisor and the referee of the thesis must be able to attend the corresponding exam.
(C5) The same teacher cannot be at the same time in more than one room.
(C6) The same teacher cannot be scheduled in different rooms subsequently.

Other important hard constraints are related with the availability of particular teachers since they may not be available all days or in some time periods only.

(C7) The teacher cannot be scheduled during time periods of his/her unavailability.

It is important to realize that the same teacher may be present in the examiner, supervisor or referee role. Actually it is a good idea to timetable students of examiners in their commissions.

(O1) Minimize the number of students which are not assigned to commissions where their supervisor or referee is the examiner.

Other good recommendation is timetabling of teachers consequently such that they do not need to come to the state examinations many times. These considerations led to the definition of two optimization criteria O2 and O3. First the number of blocks for each teacher is minimized. We consider the *block* for each teacher as the longest continuous sequence of exams where this teacher is present, i.e., no breaks are allowed in this sequence (see Table 1 where Dohnal has one block and Ráček has two blocks).

(O2) Minimize the sum of the blocks for all teachers.

This criterion would be a bit tricky for examiners since it is not necessary to group their students together. As a consequence the number of blocks always equals to one for each examiner (in his/her commission). This approach also allows an easy integration of the criterion O1 since any placement of the examiner's student to a different commission increases the number of blocks for the teacher. To conclude that, the criterion O1 is considered as a part of the criterion O2 only.

Next the number of commissions for each teacher where this teacher must be present, is minimized. These commissions of each teacher are called *assigned commissions*.

(O3) Minimize the sum of the assigned commissions for all teachers.

This criterion is somewhat involved in the primary criterion O2 but it is reasonable to consider it in different parts of the solution process. The criterion O3 can be minimized first. The goal is to place students with the same teachers to the same commission. Consequently the criterion O2 can be applied to group examinations of one teacher in one commission together.

Finally the length of examinations should be fair for all commissions.

(O4) The lengths of examinations for all commissions must be fair.

Having the same length of time periods for all exams, the fair timetable has the number of students same for all commissions or it may differ by one student at most. In manually created timetables, this criterion is often not optimally satisfied since a small unfairness is often sacrificed by a smaller number of blocks for the teachers.

3 Components of the Solver

The overall timetabling process consists of three parts implemented by three components. Initially the timetable is constructed such that all hard constraints C1-C7 are satisfied. It is not a hard problem, it is basically sufficient to add students one by one to commissions starting with assignments of students with their supervisor or referee in a commission. To avoid a high number of students of one teacher in different commissions, students are primarily assigned to commissions where students of this teacher are. If it is not possible to assign a student to the processed time period and commission due to some conflict in hard constraints, this student is assigned later. Construction of the student schedules in this stage allows an easy integration of the criterion O4 concentrated on fair lengths of timetables for all commissions. Students are just assigned to commissions to keep the same number of students at each commission or to have it different by one student only. Even more importantly this property is

kept during overall timetabling process since there are only swaps of students between two commissions or changes in the order of students in one commission.

The second component of the solver is aimed to reduce the number of assigned commissions for each teacher (the criterion O3) by the local search. The evaluation of the timetable corresponds to the sum of assigned commissions for all teachers. To evaluate student suitable for some local change, we basically take into account the sum of assigned commissions divided by the number of students for his/her supervisor and referee, i.e.,

$$\frac{\text{assigned commissions}_{\text{supervisor}}}{\text{number of students}_{\text{supervisor}}} + \frac{\text{assigned commissions}_{\text{referee}}}{\text{number of students}_{\text{referee}}} \ .$$

This is important to have a better balance among teachers — teachers with more students certainly may be assigned in more commissions. In the procedure, we subsequently take teachers one by one sorted by the number of their students decreasingly. Their worst evaluated students are subsequently chosen for the swap with other students. Simulated annealing [1] is applied to swap each chosen student with other students in committees where his/her teacher has a student. Such move is considered since it may possibly improve evaluation (the number of assigned commission) of this teacher — we basically try to group his/her students to the same commission(s), improve his/her evaluation and improve the overall evaluation of the timetable by means of the criterion O3. This procedure is repeated several times to go through each teacher repeatedly and to get improvements for him/her.

The second component aims to reduce the number of assigned committees for all teachers. However, it often happens that the main criterion O2 is worsen. Optimization of this criterion is completed by the third component of the solver. It takes the timetable of each commission one by one and optimizes number of blocks for its teachers. Students of each commission are split into several disjoint sets such that students in different sets do not share any teacher (see Table 1 where are three disjoint sets, the first one involves Lenčo Milan, the second one equals to {Škrabal Jan, Coufal Jakub} and the last one includes all other students). Within these disjoint sets containing a small number of students only, the number of blocks is improved by a tabu search [1] swapping and inserting students to different time periods. The best fit of disjoint sets to time periods is also processed to take into account block increases due to lunch break. The third component is again run repeatedly to improve the timetable iteratively.

4 Experimental Evaluation

The solver has been applied to solve two real-life data instances. The problem for Spring 2012 (Autum 2012) has 19 (11) commissions, 246 (97) students, and 195 (116) teachers. In Table 2 we compare the best timetables generated by the solver and the manually created timetables which were used during state examinations for particular semesters. We can see that the automated timetables are comparable with the timetables created manually. There is still some room for improvement since the number of blocks is still a bit better in the manual solution for the Spring 2012. On the other hand, the automated timetables are fair with respect to the length of the examination for particular commissions (the criterion O4). This is different for the manual timetables since they sometimes differ by two exams or even more. Both automated and generated timetables are available at `http://www.fi.muni.cz/~hanka/bc_state_exams`.

Table 2 Comparison of manual and automated solutions.

	Autumn 2011		Spring 2012	
	automated	manual	automated	manual
Commissions	128	121	258	259
Blocks	132	130	280	264

5 Conclusion

We have described and defined an examination timetabling problem which belongs to the class of assignment problems. This problem has been solved manually each semester and the ultimate goal of this work is the application of the automated process in practice. Future work consists in completion of the solver to handle specific exceptions and make it applicable in real life. Also it is related with the automated generation of commissions which introduces another interesting timetabling problem.

Acknowledgements We highly appreciate the support of the Grant Agency of the Czech Republic under the grant No. P202/12/0306.

References

1. Glover, F.W., Kochenberger, G.A. (eds.): Handbook of metaheuristics. Kluwer (2003)
2. Kochaniková, B.: Student timetabling for bachelor state exams at FI (in Czech) (2013). Bachelor Thesis, Masaryk University, Faculty of Informatics
3. Petr, R.: Small computer aided system for defences for final thesis and exams (in Czech) (2007). Bachelor Thesis, Masaryk University, Faculty of Informatics
4. Talbi, E.G.: Metaheuristics: From Design to Implementation. John Wiley & Sons, Inc. (2009)

MISTA 2013

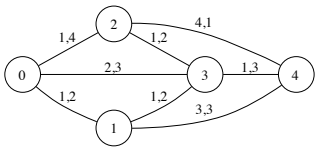
Heuristic Pruning Methods For The Robust Shortest Path Problem

Michael D Clark · Andrew J Parkes

1 Introduction

Route planning on a road network is usually associated with finding the shortest (or quickest) route between two points. It is usually modelled using the standard Shortest Path (SP) Problem on graphs with non-negative arc lengths. Given an undirected graph $G = (V, A)$ with vertex set V and arc set A, each arc $a \in A$ has an associated non-negative cost c^a which can correspond to distance or to time as desired by the user (though for simplicity we we will refer to it as a distance). The goal is to calculate the shortest path between two vertices, s and t. This problem was shown to be solvable with a complexity of $O(|V|^2)$ by Dijkstra[2], later shown to be $O(|A| + |V| \log |V|)$. This standard model relies on perfect information about the arc lengths, however in practice the real world is not perfect. If there is uncertainty in the edge lengths (*e.g.* corresponding to different traffic conditions on roads), then one way to model this is by the use of multiple scenarios. This changes the graph model $G = (V, A)$, used in the SP problem, by the addition of a 'scenario set' S, with each arc $a \in A$ has an associated length c^a_r for each scenario $r \in S$. Figure 1 shows a simple example of a five vertex, eight edge graph with two scenarios, shown as a pair on values on each edge. The graph can be in one of two states depending on the scenario: all edge lengths set to the first value of the pair or the second. The Robust Shortest Path (RSP) Problem is to find a path that minimises the value of length over all scenarios of the path

Fig. 1: A simple five node graph with 2 scenarios



Michael D Clark
University Of Nottingham
E-mail: mdc@cs.nott.ac.uk

Andrew J Parkes
University Of Nottingham
E-mail: ajp@cs.nott.ac.uk

between vertices s and t. The resulting RSP problem has been shown to be (weakly) NP-Complete with only 2 scenarios and strongly NP-Hard for unbounded scenarios [3,5]. This means that the usual SP methods such as Dijkstra's methods cannot be applied directly.

A simple Integer Programming model of the RSP can be implemented and can be used to calculate an optimal path, However, this is relatively slow even when improved using techniques such as Bender's Decomposition [4]. Hence, we implemented a branch and bound (B&B) method based on pruning the search tree in the fashion of A* search, using a best case scenario SP from the inspected vertex to the end as the Admissible Heuristic. Pruning of the search nodes is aided further with the use of a Pareto-front stored at each vertex, and corresponding to treating each scenario as creating a separate objective. When expanding the search node, if the new path dominates all previous paths visiting the incident vertex then all the store paths are pruned.

2 RSP Experiment

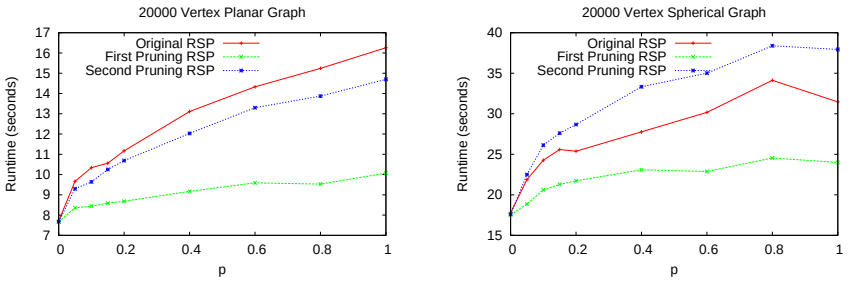
Two graph generators were constructed to produce the graphs needed for the investigation.

1. Planar Graph Generator Vertices are constructed as points in a two dimensional euclidean space. Edges are created randomly and based on manipulating the euclidean distance for the edge lengths.
2. Spherical Graph Generator Vertices are constructed randomly on the surface of a sphere by choosing a point in a cube. If the distance of the point from the centre is less than one then the point is projected onto the surface. Edge lengths are created randomly using the distance over the surface of the sphere, based on the orthodromic distance between the two vertices.

The graph generators have the ability to vary several properties: the graph density, the scenario count and the amount of variation between the scenarios, using a value $p \geq 0$. p is the proportion of the base arc length, l. The arc length for each scenario is chosen from the range $l \rightarrow l + (l * p)$.

We will present results, based on the B&B algorithm, from two lines of study. Firstly, it is well known from the work in phase transition [1], that although a problem is NP-hard it does not mean that all the instances are hard to solve, and that the hardest instances can be associated with a sharp change in the properties of the instances. We investigated using the scenario generators and found no sign of such a transition in any of the variable properties; results showed only polynomial growth of search cost with problem size and so no direct sign of the NP-hardness on average.

Secondly, we investigated whether the search costs can be reduced by pruning the graph vertices based on using 'interpolated scenarios' in which edge lengths are just a linear combination of the given scenarios. The SP is calculated for each of multiple interpolated scenarios, and a graph vertex is pruned if the vertex is not used in any of the resulting SPs. The RSP algorithm is applied to the reduced graph (pRSP) and the resulting path and its associated costs are compared to the RSP from the original graph. The initial investigation used only two scenarios and varied p for two interpolated sets.



(a) Planar Graph (b) Spherical Graph

Fig. 2: Runtime Comparison of the Interpolation Methods

3 Conclusion

The initial results have shown that the RSP obtained from the reduced graph uses much less space and time, as a close to linear growth rate for the search tree size; figure 2 clearly shows the runtime results of the RSP and the reduced RSP together, it can be seen that the first interpolation set has the faster runtime and lowest growth rate with respect to p. Furthermore, table 1 shows that the worse case path cost deterioration is less than 3.85% for 20000 vertices and less than 0.33% in the median case. The insights gained from the interpolation investigation show that *NP-Hard* problems in the average case can be evaluated in polynomial time. These insights could be adapted for the robust Vehicle Routing Problem (VRP) and could be applied to similar poly-time scheduling problems where the robust version is *NP-Hard*. Further investigation into whether the initial results hold for $N > 2$ scenarios has been undertaken and the results will be shown and discussed elsewhere.

	20000 Vertex, median(max)			
	Planar Graphs		Spherical Graphs	
p	I_1 (%)	I_2 (%)	I_1 (%)	I_2 (%)
0.0	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
0.2	0.0 (0.69)	0.0 (0.62)	0.0 (1.12)	0.0 (0.56)
0.4	0.08 (1.42)	0.20 (1.68)	0.0 (1.96)	0.0 (2.12)
0.6	0.27 (1.90)	0.06 (1.62)	0.09 (2.98)	0.0 (2.44)
0.8	0.26 (2.84)	0.18 (2.50)	0.33 (1.82)	0.30 (2.97)
1.0	0.19 (3.85)	0.18 (3.85)	0.24 (2.76)	0.0 (2.22)

Table 1: Comparison of the Interpolation Path Costs

References

1. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In: Proceedings of the 12th International Joint Conference on Artificial intelligence - Volume 1, IJCAI'91, pp. 331–337. Morgan Kaufmann Publishers Inc. (1991)
2. Dijkstra, E.W.: A note on the two problems in connection with graphs. Numerische Mathematik **1**, 269–271 (1959)
3. Kouvelis, P., Yu, G.: Robust Discrete Optimization and Its Applications (Nonconvex Optimization and Its Applications). Springer (1996)
4. Montemanni, R., Gambardella, L.M.: The robust shortest path problem with interval data via Benders decomposition. 4OR **3**, 315–328 (2005)
5. Yu, G., Yang, J.: On the robust shortest path problem. Computers and Operations Research **25**, 457–468 (1998)

MISTA 2013

A Grouping Hyper-heuristic for Graph Colouring

Anas Elhag · **Ender Özcan**

1 Introduction

Many well known combinatorial optimisation problems, such as data clustering, timetabling, and graph colouring are classified as *grouping problems* [6], requiring the same common task of partitioning a set of objects U into a collection of mutually disjoint subsets u_i of U such that each object is in exactly one subset, as follows:

$$\begin{aligned} \bigcup u_i &= U \quad \forall i \\ u_i \bigcap u_j &= \emptyset \quad \forall i, \forall j \quad \textit{where } i \neq j \\ u_i &\neq \emptyset \quad \forall i \end{aligned} \tag{1}$$

Mostly, the overall fitness f of a grouping problem solution $U_g = \{u_1, ..., u_i, ..., u_k\}$ can be measured using an evaluation function which adds up the partial contribution from each group as in Equation 2.

$$f(U_g) = \sum_{i=1}^{k} f(u_i) \tag{2}$$

Assuming that each colour represents a group, then the graph colouring problem [5] can be formulated as a grouping problem in which the task is to assign each vertex of an undirected graph to a group, such that no two connected vertices are in the same group, with the goal of minimising the number of groups. Formally, given a graph $G = (V, E)$ with vertex set V and edge set E, and given an integer k, a *k-grouping* of G is a function $u_i : V \rightarrow 1, ..., k$, where u_i of a vertex x is the group of x. If two connected vertices x and y are in the same group u_i, x and y are conflicting vertices, and the edge $E_{x,y}$ is called a conflicting edge. If there are no conflicting edges, then the groups are all independent sets and the *k-grouping* is valid. The graph colouring problem is to determine the minimum integer k (the chromatic number of $G - \chi(G)$) such that there exists a legal *k-grouping* of G.

A. Elhag and E. Özcan
University of Nottingham, School of Computer Science
Jubilee Campus, Nottingham NG8 1BB UK
Tel.: +44 (115) 95 15544
Fax: +44 (115) 846 7877
E-mail: {axe, exo}@cs.nott.ac.uk

Clearly, not all groupings are feasible for all grouping problems, since different grouping problems impose different constraints and so they introduce different objectives. For example, in graph colouring, two connected nodes must not be placed into the same group.

Also, a common feature of the grouping problems as defined is that they require the number of groups k to be minimized as well as the combined fitness of the groups $f(U_g)$. Consequently, grouping problems can be considered to be multi-objective optimisation problems [10]; or more precisely, bi-objective optimisation problems. In the graph colouring problem, the minimum number of colours causing the least conflicts is searched. These objectives are conflicting objectives; i.e optimising one of them will cause the other to deteriorate. Solving a grouping search problem will, hopefully, yield a set of optimal solutions at the end of the search process. Each one of these solutions is better than the others in terms of one of the objectives, and worse in terms of the other. This set of optimal solutions is known as the *pareto-optimal front*. Traditionally, the pareto-optimal front is approximated by a set of what is called non-dominated solutions. A solution U_1 dominates another solution U_2 if (i) U_1 is no worse than U_2 in anyone of the objectives, and (ii) U_1 is better than U_2 in at least one objective.

Hyper-heuristics are methodologies that select or generate heuristics during the search process for solving computationally hard problems [4, 8]. A software framework, known as HyFlex, has been recently developed to support the design of cross-domain heuristic search methods particularly hyper-heuristics [2]. The Java version of HyFlex implementing six different problem domains was recently used at the first Cross-Domain Heuristic Search Challenge (CHeSC 2011) [3], which then became a benchmark for selection hyper-heuristics. Each problem domain implementation came with its specific set of low level heuristics. In this study we are aiming to achieve a different level of generality by designing a selection hyper-heuristic framework that is provided with a fixed set of low level heuristics which can be used for solving all grouping problems. The low level heuristics are designed based on the representation suggested as part of the Genetic Grouping Algorithm (GGA)[6]. Our aim is not to beat the state of the art techniques which are designed and tuned for a particular problem domain. We aim to provide the same selection hyper-heuristic framework based on a single point based search approach for all grouping problems. The initial tests using our framework is performed on instances of the graph colouring problem.

2 Experimental Results

Instances from the DIMACS benchmark suite are used for the experiments. For each instance, a range for the suitable number of groups k is selected around the known best number. Then, an initial pareto front is created by building a random solution for each value of k. Each of these solutions is then iteratively improved using a selection hyper-heuristic which perturbs the current solution generating a new one using a chosen low level heuristic and then decides whether to accept or reject the new solution. At the end of the search, we use a method known as *the elbow criterion* to determine the best point in the pareto front. Seven different perturbative low level heuristics were implemented. The main objective of these low level heuristics is to make modifications on the current solutions, such as dividing a group, merging two groups or swapping items between groups. Different hyper-heuristics were tested during the experiments. The experiments were carried out on a 3.6GHz Intel Core i7-3820 Windows 7 machines with a memory of 16GB. Each experiment is repeated 31 times and a run is terminated after 600 seconds.

The results of the experiments show that our proposed framework successfully finds the best solutions for all the data sets. In the overall, a modified version of the (GIHH)[1] hyper-heuristic [7] was found to be the best among the tested hyper-heuristics in terms of improving the pareto fronts as well as finding the best solutions. Also, our framework pushes the initial pareto fronts further than a previously proposed framework that uses a different grouping representation known as linear linkage encoding (LLE) [1]. Yet, the same best-of-runs results are obtained. Figure 1 shows a comparison between the initial and the final pareto front obtained by GIHH for data set DSJC125.5. Figure 2 shows a comparison between the ratios of calls, acceptance and acceptance for best made to each low level heuristic by GIHH hyper-heuristic on DSJC125.5 data set. Table 1 shows a comparison between the best results obtained by our approach (GGA-GIHH) to those obtained by Greedy Partition Crossover Lowest Index (GPX-LI), Greedy Partition Crossover Cardinality Based (GPX-CB) and Lowest Index Max Crossover (LIMX) proposed in [9]. Initial experiments show that our best-of-runs results are competitive with the previously proposed approaches.

More details of the multi-objective framework hyper-heuristic approach as well as more results on some other grouping problem domains will be provided at the conference.

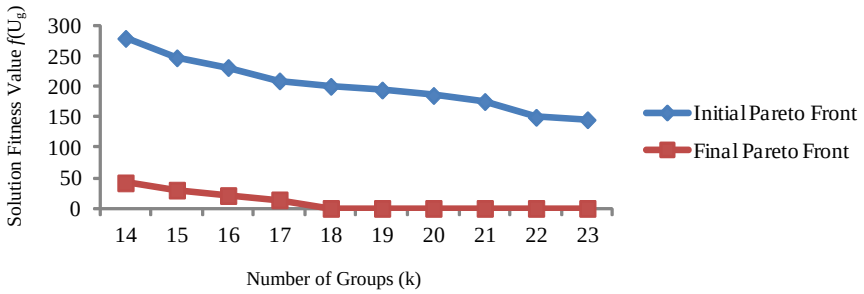


Fig. 1 Comparison between an initial and a final pareto fronts obtained by the AdapHH on DSJC125.5 data set.

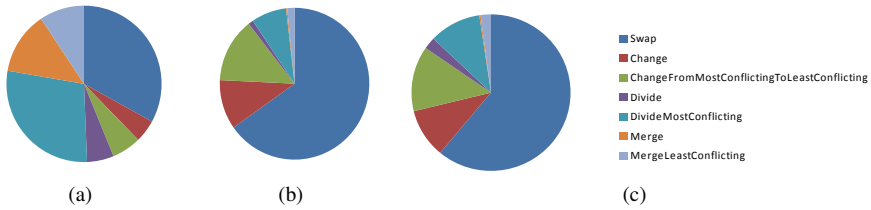


Fig. 2 The ratios of low level heuristics (a) selected at 2,465,530 decision points, (b) accepted considering 910,064 (out of 2,465,530) times, and (c) yielding improvement in the best solution considering 288,151 (out of 2,465,530)by AdapHH on DSJC125.5.

[1] http://www.code.google.com/p/generic-intelligent-hyper-heuristic/downloads/list

Table 1 Best results comparison

Instance	$\chi(G)$	GGA-GIHH	LIMX	GPX-LI	GPX-CB
DSJC125.5	?	**18**	**18**	**18**	**18**
DSJC125.9	?	**44**	**44**	**44**	**44**
DSJC250.1	?	**9**	**9**	**9**	**9**
DSJC250.5	?	**30**	31	31	31
DSJC250.9	?	**74**	75	75	**74**
DSJC500.1	?	**14**	**14**	**14**	**14**
le450_15a	15	**15**	16	16	16
le450_15b	15	**15**	16	16	16
le450_15c	15	**15**	23	23	23
le450_15d	15	**15**	23	23	23
le450_25a	25	**25**	**25**	**25**	**25**
le450_25b	25	**25**	**25**	**25**	**25**
le450_25c	25	**25**	28	28	28
le450_25d	25	**25**	28	28	28
Wins/Draws		**7/7**	0/6	0/6	0/7

References

1. Birben, M.: Solving grouping problems using a selection hyper-heuristic framework based on linear linkage encoding. Tech. rep., Yeditepe University, Computer Engineering (2011)
2. Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vazquez-Rodriguez, J.: Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In: Proceedings of the Multidisciplinary International Scheduling Conference (MISTA09), pp. 790–797 (2009)
3. Burke, E., Gendreau, M., Hyde, M., Kendall, G., McCollum, B., Ochoa, G., Parkes, A., Petrovic, S.: The cross-domain heuristic search challenge - an international research competition. In: X. Yao, C.A.C. Coello (eds.) Proceedings of Learning and Intelligent Optmization (LION5), *Lecture Notes in Computer Science*, vol. 6683, pp. 631–634 (2011)
4. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. Tech. Rep. NOTTCS-TR-SUB-0906241418-2747, University of Nottingham, School of Computer Science (2010)
5. Chiarandini, M., Stützle, T.: An analysis of heuristics for vertex colouring. In: P. Festa (ed.) Experimental Algorithms, Proceedings of the 9th International Symposium, (SEA 2010), *Lecture Notes in Computer Science*, vol. 6049, pp. 326–337. Springer (2010)
6. Falkenauer, E.: Genetic Algorithms and Grouping Problems. John Wiley & Sons, Inc., New York, NY, USA (1998)
7. Misir, M., Verbeeck, K., Causmaecker, P.D., Berghe, G.V.: An intelligent hyper-heuristic framework for chesc 2011. In: Y. Hamadi, M. Schoenauer (eds.) LION, *Lecture Notes in Computer Science*, vol. 7219, pp. 461–466. Springer (2012)
8. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive survey of hyperheuristics. Intelligent Data Analysis **12**(1), 3–23 (2008)
9. Ülker, O., Özcan, E., Korkmaz, E.E.: Linear linkage encoding in grouping problems: Applications on graph coloring and timetabling. In: International Conference on the Practice and Theory of Automated Timetabling (2006)
10. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: Empirical results. Evol. Comput **8**(2), 173–195 (2000)

MISTA 2013

The stochastic resource-constrained project scheduling problem under different confidence levels

Patricio Lamas · Erik Demeulemeester

1 Introduction

The resource-constrained project scheduling problem (RCPSP) aims to determine for each activity its start and finish time in order to minimize the project duration. Each activity is defined by a duration, a renewable resource consumption and a set of precedence activities. The total availability of the renewable resource type is known. The classical version of this problem assumes that the project is executed in a deterministic environment, i.e. activity durations, renewable resource consumptions and renewable resource availabilities are deterministic parameters. In practice, however, projects are subject to considerable uncertainty (e.g. bad weather conditions, accidents, resource breakdowns, etc.). Thus, the assumption of deterministic parameters is not accurate in a real life context.

We propose a proactive scheduling method based on a chance-constrained programming (CCP) formulation of the RCPSP, where the activity durations are random variables. The objective is to minimize the project makespan, while the deviation between the real executed and proactive activity starting times is taking into consideration. Previous work related to the proactive stochastic RCPSP are [4–6,8,9].

2 Problem statement

Given are a set of activities $V = \{A_0, \ldots, A_{n+1}\}$. A_0 and A_{n+1} are dummy activities that represent the start and the end of the schedule respectively. p_i (random variables) represent the duration of activity A_i, with $p_0 = p_{n+1} = 0$. The set of precendence relations is given by E (if A_i precedes A_j then $\left(A_i, A_j\right) \in E$). The set of resources is represented by $R = \{R_1, \ldots, R_k\}$. B_k is the availability of resource R_k and b_{ik} is the

Patricio Lamas
Research Center for Operations Management, KU Leuven (Belgium)
E-mail: Patricio.LamasVilches@kuleuven.be

Erik Demeulemeester
Research Center for Operations Management, KU Leuven (Belgium)
E-mail: Erik.Demeulemeester@kuleuven.be

consumption of resource R_k during the execution of activity A_i. F represents a forbidden set, which is a subset of unrelated activities (i.e., whithout precedence relations) that cannot be scheduled in paralell because of resource limitations ($\sum_{A_i \in F} b_{ik} > B_k$). Φ is the set of all minimal forbidden sets. Finally, $1-\alpha$ is a confidence level defined by the decision maker (typically 0.99 or 0.95).

Defining the binary variable x_{ij} which takes value 1 if activity A_j starts after the completion of activity A_i, and 0 otherwise, and the non-negative variable S_i representing the starting time of activity A_i, the CCP-RCPSP can be formulated as follows (see [3]):

$$\min S_{n+1} \tag{1}$$

subject to

$$x_{ij} = 1 \quad \forall (i,j) \in E \tag{2}$$

$$x_{ij} + x_{ji} \leq 1 \quad \forall (i,j) \in V^2 \tag{3}$$

$$\sum_{(i,j) \in F^2} x_{ij} \geq 1 \quad \forall F \in \Phi \tag{4}$$

$$P\left(S_j - S_i \geq \left(1 - x_{ij}\right) M + p_i \quad \forall (i,j) \in V^2\right) \geq 1 - \alpha \tag{5}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in V^2 \tag{6}$$

$$S_i \geq 0 \quad \forall i \in V \tag{7}$$

Constraints (2) are the precedence constraints. Constraints (3) ensure that no cycles are in the schedule. Constraints (4) represent the resource availability contraints. Contraint (5) states that the probability that starting times of activity A_i and A_j are feasible is larger than the confidence level $1-\alpha$.

3 Solution method

Because of constraint (5) the CCP-RCPSP cannot be directly input in a MIP solver. In [1] a sample average approximation (SAA) method is proposed. In a first step a set W of the random variables must be created. Then, the problem is to find a solution that is feasible for $\lceil |W| \cdot (1-\alpha) \rceil$ scenarios.

Defining p_{iw} as the duration of activity A_i for scenario w and the binary variable y_w which takes value 1 if the solution is not feasible for scenario w and 0 otherwise, constraint (5) can be replaced by:

$$S_j - S_i \geq \left(1 - x_{ij}\right) M + \left(1 - y_w\right) p_{iw} \quad \forall (i,j) \in V^2, w \in W \tag{8}$$

$$\sum_{w \in W} y_w \leq \lfloor |W| \cdot \alpha \rfloor \tag{9}$$

$$y_w \in \{0,1\} \quad \forall w \in W \tag{10}$$

The formulation defined by (1)-(4),(6)-(10) can be strengthened by using the valid inequalities presented in [2].

4 Results

The presented method has been tested for 480 instances belonging to PSPLIB [7]. The stochastic activity durations are assumed to be random variables following a discretized *Beta* distribution, with a mean equal to the activity duration values in PSPLIB. The problem was solved for 11 different confidence levels $(1-\alpha)$= 0.99,0.95,0.9,0.8,...,0.1. A trade-off curve between makespan and confidence level was obtained. Finally, the method was compared with the starting time criticality heuristic (STC) presented in [10]. The results of our method dominate the ones obtained by the STC heuristic in terms of quality. However, the execution times for our method are larger.

References

1. Luedtke, James and Ahmed, Shabbir, A sample approximation approach for optimization with probabilistic constraints, SIAM Journal on Optimization, 19, 674-699 (2008)
2. Luedtke, James and Ahmed, Shabbir and Nemhauser, George L., An integer programming approach for linear programs with probabilistic constraints, Mathematical Programming,122,247-272 (2010)
3. Alvarez-Valdes, Ramon and Tamarit, Jose Manuel, The project scheduling polyhedron: dimension, facets and lifting theorems,European Journal of Operational Research, 67,204-220,(1993)
4. Filip Deblaere, Erik Demeulemeester, and Willy Herroelen, Proactive policies for the stochastic resource-constrained project scheduling problem,European Journal of Operational Research,214,308-316,(2011)
5. Filip Deblaere, Erik Demeulemeester, Willy Herroelen and Stijn Van de Vonder. Robust resource allocation decisions in resource-constrained projects, Decision Sciences, 38,5-37,(2007)
6. Willy Herroelen and Roel Leus. The construction of stable project baseline schedules. European Journal of Operational Research, 156, 550-565, (2004)
7. Rainer Kolisch and Arno Sprecher, Psplib-a project scheduling problem library: Or softwareorsep operations research software exchange program, European Journal of Operational Research, 96, 205-216, (1997)
8. Olivier Lambrechts, Erik Demeulemeester and Willy Herroelen. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. J. Scheduling, 11, 121-136, 2008.
9. Olivier Lambrechts, Erik Demeulemeester, and Willy Herroelen. Time slack-based techniques for robust project scheduling subject to resource uncertainty, Annals OR, 186, 443-464, 2011.
10. Stijn Van de Vonder, Erik Demeulemeester and Willy Herroelen. Proactive heuristic procedures for robust project scheduling: An experimental analysis, European Journal of Operational Research, 189, 723-733, 2008.

MISTA 2013

Initial Results on Fairness in Examination Timetabling

Ahmad Muklason · Andrew J. Parkes · Barry McCollum · Ender Özcan

1 Introduction

Examination timetabling problems have attracted many researchers during the last couple of decades, and especially since the work of Carter [2], and Carter, Laporte and Lee [3]. The problems are NP-complete and challenging, and so have nurtured different approaches and techniques; for a recent survey see [10].

In standard formulations the quality of a solution, from the student perspective, is given by an objective function which is a simple weighted sum of penalties for the timetable of each student. For example, in the classic Toronto benchmarks [3], the penalty per student is designed to reflect the natural desire of students that their exams do not take place too close together in time. Hence, minimising the objective is intended to maximise the average student satisfaction with the personal spread of examinations; however, it does not do anything to ensure equality between students. Students may consider the assessments to be unfair if some students have well spread out exams (small penalty) whilst others have many exams close together (large penalty). We believe that it is reasonable that overall student satisfaction could also be improved by increasing the fairness of treatment between students. In this paper, we make preliminary investigations of how to increase such fairness.

For general background, the common sense definitions of fairness in political science and political economics are discussed in [6] which defines fairness as an allocation where "no person in the economy prefers anyone else's consumption bundle over his own." In general resource allocation, there are two well-accepted and common notions of fairness criteria: *max-min* fairness and *proportional* fairness. *Max-min* fairness allows to say an allocation is fairer than another allocation but does not measure how much fairer [11], whilst *Proportional* fairness is quantitative measure of fairness (see [1] for details). Fairness has been studied before in combinatorial optimisation problems; for

Ahmad Muklason, Andrew J. Parkes, Ender Özcan
School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, United Kingdom
E-mail: {abm, ajp, exo}@cs.nottingham.ac.uk

Barry McCollum
School of Computer Science, Queen's University, Belfast, BT7 1NN, UK.
E-mail: b.mccollum@qub.ac.uk

example, flight landing scheduling [13] and nurse rostering [4,12]. However, to the best of our knowledge, there have only been a few works dealing with fairness in educational timetabling. The most related prior work is for course timetabling by [9], and in which two formulations based on max-min fairness and the "Jain's Fairness Index" [7] are proposed. Here, rather than use max-min methods, in order to improve fairness in examination timetabling, we study the effects of modifying the objective function.

2 The Modified Objective Function

In order to take fairness into account, the basic idea in this preliminary investigation is by modifying the objective function. Instead of using standard linear summation (LS) objective function, a modified objective functions (MS) is studied.

$$LS = \sum_{s=1}^{S} P_s \tag{1}$$

$$MS = \sum_{s=1}^{S} f(P_s) \tag{2}$$

Where,

$$P_s = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} P(s,i,j) \tag{3}$$

$$P(s,i,j)) = \begin{cases} W_{|t_j - t_i|} & \text{if student s takes exams i and j} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Here, S is the total number of students, P_s is total penalty of the s'th student, N is the number of examinations, $P(s,i,j)$ is the penalty of the s'th student taking examination i and j, and t_i is a timeslot where an examination i is scheduled. $W_{|t_j - t_i|}$ is the weight whenever a student sits two examinations that are scheduled $\mid t_j - t_i \mid$ apart. The penalty weight, $W_{|t_j - t_i|}$ is calculated as $2^{5-|t_j - t_i|}$ where, $\mid t_j - t_i \mid \in \{1,2,3,4,5\}$. $W_{|t_j - t_i|}$ is equal to 0 if $\mid t_j - t_i \mid > 0$..

The function $f(P_s)$ is the function of total individual student penalty and that can be used to encourage fairness by using a non-linear function that increases the penalty on larger initial penalties. In this study, we considered the fairly standard case of $f(P_s)$ as Sum of Squares (SoS), but also the more general Sum of Powers (SoP):

$$\text{SoS}: \quad f(P_s) = P_s^2 \tag{5}$$

$$\text{SoP(p)}: \quad f(P_s) = P_s^p \tag{6}$$

3 Experimental Methods and Results

The examination timetabling solver used for this paper is a standard two phase algorithm, consisting of a sequential heuristic construction method followed by an improvement phase based on the great deluge algorithm [5]. We use Jain's Fairness Index (JFI) [7] to measure the fairness of the solutions. Suppose a solution from a timetabling solver is T, the JFI of the solution is defined as follows in terms of the penalties, P_s, for the individual students:

$$JFI(T) = \frac{\left(\sum_{s=1}^{S} P_s\right)^2}{\left(S * \sum_{s=1}^{S} (P_s)^2\right)} \tag{7}$$

Note that the JFI value is bounded between 0 and 1, with larger values implying the solution becomes more fair. A value of 1 corresponds to complete fairness.

Table 1 Experimental result: solutions' fairness produced from LS, SoS, and SoP objective function

Inst.	LS		SoS		SoP	
	MEAN	JFI	MEAN	JFI	MEAN	JFI
CAR91	5.46	0.33	5.56	0.35	8.33	0.39
CAR92	4.96	0.29	4.83	0.31	6.74	0.35
EAR83	39.8	0.82	39.42	0.84	47.88	0.86
HEC92	11.24	0.49	11.34	0.52	15.32	0.58
KFU93	16.25	0.54	16.26	0.56	20.07	0.63
LSE91	13.36	0.53	13.42	0.57	17.16	0.63
PUR93	5.84	0.35	6.06	0.38	8.62	0.44
RYE92	9.82	0.37	9.7	0.39	16.21	0.44
STA83	163.08	0.91	163.12	0.91	168.01	0.93
TRE92	8.82	0.44	8.93	0.47	11.16	0.49
UTA92	4.23	0.24	4.31	0.26	5.22	0.29
UTE92	26.54	0.79	26.51	0.8	31.59	0.81
YOR83	39.12	0.75	38.8	0.77	48	0.77

In this preliminary study, the experiment is carried out under Carter dataset [3]. In the experiments, for each instance in the dataset we conduct 20 test runs. The results are given in Table 1, and compares the mean penalty cost (MEAN) and Jain's Fairness Index (JFI) of solutions resulting from three different objective functions, specifically, linear summation (LS), Sum of Squares (SoS) and Sum of Powers (SoP).

It can be seen from Table 1 that both SoS and SoP objective function can produce fairer solutions for all problem instances, though sometimes with some loss of quality of the standard linear sum. Not surprisingly, SoP can produce fairer solution than SoS, however, it was somewhat surprising that we had to go as high as the 16th power to get a significant effect, and that the common SoS approach was not always sufficient to improve the fairness significantly. However, SoS compensates with a smaller increase in the average linear penalty cost than with SoP, and even produced smaller average such as in instance UTE92, TRE92, and EAR83 – though this may just be an artefact of the search algorithm used.

4 Conclusions

This paper has presented a preliminary study which attempts to improve fairness in examination timetabling solutions. A modified objective function is proposed instead of traditional linear summation of penalties, and the experimental results show this approach produced fairer solution for almost all problem instances. However, not unexpectedly, it also shows that fairer solution can also cause a slightly worse overall average quality of solution. Naturally, the problem is one of multi-objective with a balance to be struck between quality and fairness. This work encourages future investigation of search methods for finding fairer solutions. In future studies, in addition to the student perspective, it may also important to take into consideration the quality and fairness of solution from the perspective of other stakeholders such as invigilators, lecturers, and estates, and also in more realistic formulations [8].

References

1. Dimitris Bertsimas, Vivek F. Farias, and Nikolaos Trichakis. The price of fairness. *Operations Research*, 59(1):17–31, 2011.
2. Michael W Carter. A survey of practical applications of examination timetabling algorithms. *Operations Research*, 34(2):193–202, March 1986.
3. Michael W. Carter, Gilbert Laporte, and Sau Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47(3):373–383, 1996.
4. A. A. Constantino, D. Landa-Silva, E. L. de Melo, and W. Romao. A heuristic algorithm for nurse scheduling with balanced preference satisfaction. In *Computational Intelligence in Scheduling (SCIS), 2011 IEEE Symposium on*, pages 39–45, 2011.
5. Gunter Dueck. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(1):86–92, 1993.
6. Duncan K. Foley. Resource allocation in the public sector. *Yale Econonomic Essays*, 7(1):45–98, 1967.
7. Rajendra K Jain, Dah-Ming W. Chiu, and William R Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. *ACM Transaction on Computer System*, 1984.
8. Barry McCollum, Paul McMullan, Andrew J. Parkes, Edmund K. Burke, and Rong Qu. A new model for automated examination timetabling. *Annals of Operations Research*, 194(1):291–315, 2012.
9. Moritz Mühlenthaler and Rolf Wanka. Fairness in academic timetabling. In *Proc. 9th Int. Conf. on the Practice and Theory of Automated Timetabling (PATAT)*, pages 114–130, 2012.
10. R. Qu, E.K. Burke, B. McCollum, L.T.G. Merlot, and S.Y. Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1):55–89, 2009.
11. J. Rawls. *A Theory of Justice*. Belknap Press, 1971.
12. P. Smet, S. Martin, Ouelhadj Djamila, E. Özcan, and G. Vanden Berghe. Investigation of fairness measures for nurse rostering. In *9th International Conference on the Practice and Theory of Timetabling (PATAT 2012)*, 2012.
13. M. J. Soomer and G. M. Koole. Fairness in the aircraft landing problem. In *Anna Valicek Competition 2008*, 2008.

MISTA 2013

Innovative teaching and the death of the utilisation survey

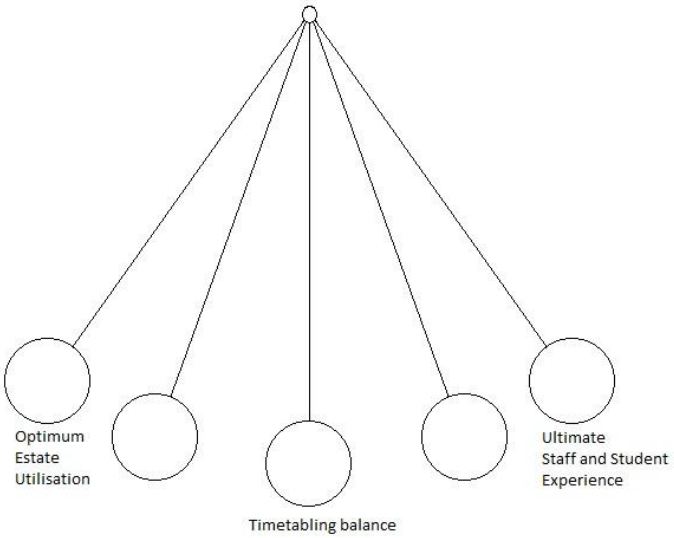
M. Earl

Introduction

The academic timetable is the product of scheduling the delivery requirements of academic modules whilst considering student intake, progression and modular choice, alongside academic staff constraints and estate resource. The two main timetable objectives are experience (for both staff and students) and estate utilisation. If the timetabling focus is purely on experience, then the rooms allocated will be through preference or location regardless of size. Conversely, if the goal is maximum utilisation then classes would be scheduled to best fit regardless of time or location. However, as McCollum [1] explains, 'an overriding principle should be that, in striving towards the optimisation of space, teaching quality must not be allowed to suffer.' Therefore, depending on the current economic or sector requirements, the goal of the timetabling team is either to find the balance or look to produce timetables which aim to satisfy one objective whilst minimising impact on the other.

These contradicting objectives can be represented by a pendulum with ultimate student/staff experience at one end and optimum estate utilisation at the other, as represented by the schematic diagram in figure 1 below:

Figure 1: The timetabling pendulum



The role of the timetabler is to be both proactive and reactive and, in the last 2 decades, timetablers across the sector have worked towards this balance between experience and utilisation. In 2012, for example, the timetabling reaction to the rise in student fees led to a swing of the pendulum further towards student experience than ever before. The next major pendulum swing will be working towards 2020 and the university carbon agenda where all institutions are required to cut carbon emissions by 43% [2]. This will result in poor

M Earl
Timetabling Manager, Facilities Management,Loughborough University
Loughborough
E-mail: m.earl@lboro.ac.uk

performing buildings being lost and the timetabling pendulum will inevitably swing back towards utilisation.

So, how do we measure utilisation? In 1996, a paper entitled 'The Management of Space in Higher Education Institutions in Wales' [3] highlighted the importance of effective space management and introduced the utilisation survey formula for measuring the efficiency of university space usage as:

$$\frac{\%\ \text{frequency} \times \%\ \text{occupancy}}{100} = \text{space utilisation rate} \quad [3]$$

This equation shows *frequency* (how often a room is used) and *occupancy* (how full a room is once in use) given equal weighting to the overall utilisation percentage [3,4]. However, much has changed since 1996, with one important factor being the methods of teaching given the rapid introduction of new educational technologies. Chalk boards and overhead projectors are becoming extinct and new technologies are being introduced. Visualizers, split screen projection, plectrum tables, lecture capture and, in some cases, innovative classrooms developed for specific styles are being installed all across the sector. It would not be possible to install all of these modern tools into every teaching room, which consequently influences room usage. Even if this were possible, by the time all installations were complete there would be more, newer, products on the market.

In addition, the delivery of lectures is required to evolve with technology. If a lecturer has amended their teaching style from using chalk boards to innovative split screen projection, this would be seen as a natural progression into modern teaching. In this example, what would happen if the following year's student intake meant that the split screen room was far too big for the module? Does the lecturer need to amend their pedagogic style annually according to recruitment? Or will there ever be a point in the process of scheduling academic events when style overrules capacity?

Through the exploration of the occupancy element within the utilisation formula, this paper will analyse whether specific modular requirements can evoke a change to the formula which evaluates the best use of the estate. Therefore, will the utilisation survey also need to evolve to factor in a new variable: technology? The proactive approach to timetabling research would therefore be to revisit and potentially reassess the utilisation formula now, while all eyes are still on experience, in preparation for the next major pendulum swing when the timetabling role will once again be assessed primarily on the utilisation percentage.

Methods of timetable evaluation

The two, often contradictory, timetable objectives are student experience and utilisation, both of which need to be considered whilst scheduling. As Pryzibilla [5] states, 'the institution is interested in having a timetable that is efficient and effective, and that makes good use of resources (space, equipment, teaching and ancillary staff)' Later in the academic year, both of these objectives are evaluated; utilisation through the utilisation survey and student experience through the National Student Survey (NSS). Each of these surveys solely evaluates one objective, without any consideration of the other. It is interesting to note that, whilst both utilisation and experience are to be considered throughout the scheduling process, there is no timetable assessment that considers both objectives at the same time.

To get a better representation of timetabling success, the utilisation survey needs to take into consideration the timetabled events that have been scheduled without best fit as first priority. In order to do this we must first analyse the current utilisation survey to understand how the formula is constructed.

Understanding the utilisation survey

Annual utilisation surveys are a sector requirement but the survey as it stands is too blunt an instrument. The resulting utilisation percentage is made up of the two factors; frequency and occupancy. This does not tell the story of what external factors, and timetabling decisions, led to the end result. The utilisation survey outcome is expressed as a percentage; the HEFCE benchmark for good estate utilisation [4] is greater than 35%, fair is between 25 – 35%, with poor being 25% or less based on the survey results. So, satisfactory utilisation can be as low a percentage as 25%. Such a low figure is unhelpful in building an accurate picture of utilisation as, for those without full knowledge the way it is calculated, there is a perception that effective utilisation would be much closer to 100%.

Estate regeneration decisions are made based on utilisation statistics and without a full understanding of what the individual utilisation components represent the resulting utilisation figure may give an indication that less space is being used than the actual figure. For example, if a building reached a frequency of 70%, and the occupancy level matched the 70% it would be perceived that the utilisation percentage would be equally high. Because of the multiplication of these percentages this is not the case:

Frequency X Occupancy = Utilisation
70% X 70% = **49%**

This is represented in Table 1 below:

Table 1: Utilisation rates

		Frequency									
		10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
Occupancy	**10%**	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
	20%	2%	4%	6%	8%	10%	12%	14%	16%	18%	20%
	30%	3%	6%	9%	12%	15%	18%	21%	24%	27%	30%
	40%	4%	8%	12%	16%	20%	24%	28%	32%	36%	40%
	50%	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%
	60%	6%	12%	18%	24%	30%	36%	42%	48%	54%	60%
	70%	7%	14%	21%	28%	35%	42%	49%	56%	63%	70%
	80%	8%	16%	24%	32%	40%	48%	56%	64%	72%	80%
	90%	9%	18%	27%	36%	45%	54%	63%	72%	81%	90%
	100%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%

The utilisation in this scenario is **49%**. This would usually be perceived to be not even half full, which is untrue. The rooms in this example are being used 70% of the time and unless recruitment falls into line with room sizes then the occupancy component of the equation will affect the end percentage. This example shows that the utilisation percentage is not always the full picture.

Innovative teaching

Teaching styles are evolving as a result of new educational innovations. This, combined with the rise in student fees and the 'Apple generation' of students, has meant that there has been a fast shift towards an expectation of the latest technological provision. Academia is thus being influenced by external factors and must now consider this, extra, competitive, variable of innovation.

As we can see, new technologies introduce an additional component to the scheduling process. The new equipment is specific to certain university spaces, so, from a timetabling perspective, innovation has become an added variable to factor in when scheduling. To revisit the example discussed previously, a lecturer has adapted their teaching style in order to deliver lessons on split-screen digital projection, where one screen will run a presentation and other running a demonstration through a visualizer. The timetable will endeavour to schedule the lecturer's session into a space that both provides the innovative kit whilst also achieving good utilisation (alongside the other various timetable considerations such as staff/student clashes, room availability, travel time). As the timetable is considering more factors while scheduling, shouldn't the assessment of the timetable process also consider those factors?

The death of the current utilisation survey

The original survey is almost twenty years old and, since its development, not only have teaching methods changed but also timetabling systems and methods of data collection have advanced. Optimum timetable decisions are made throughout the scheduling process, with pedagogic requirements factored against a range of constraints. As we have seen, the main considerations when scheduling timetable events are best fit and location. The survey requirements are to inspect every room, on the hour, to identify first whether the room is in use, and if in use, to secondly count how many people are using the room. If best fit was the only timetabling consideration then the survey would indicate an accurate representation of utilisation and for many scheduled events this is the case.

However, if, for example, a module of 40 students is scheduled into a 120 capacity room with specialist kit such as split screen projection, then occupancy would currently be calculated at 30%. This figure is not representative of the initial decision made during the scheduling process. The proposed occupancy in this specific event, in line with student experience, would be 100%. This is because the technological requirements had meant that it was necessary to schedule that event into that room. Therefore, the adaptation to the utilisation survey for specialist events would mean that the surveyed occupancy component should be divisible by the number of students registered, and not the size of the room.

Therefore, should a module require specialist kit for a particular event which would contradict the set principles of utilisation assessment, then that event would be flagged at source in the timetable system, then scheduled and roomed accordingly. The flag against the specialist event is then identified when the utilisation data is compiled and the assessment criteria adapted to suit the conditions of the booking. This added flag to the event in the timetable system would enable the incorporation of the student experience factor into the utilisation survey and would build the relationship between scheduling and surveying.

Where do we go from here?

From a modern scheduling perspective, there is a point at which teaching style will need to overrule the room capacity. If decisions such as this could be factored into the timetable system then this could then carry through to the utilisation survey and would provide a more accurate representation of the survey in the real world.

The main components of the utilisation survey would then be:

- *Frequency*; how often a room is used
- *Occupancy*; the number of people in a room divided by room capacity for general events, or the number of people in the room divided by the module size for specialist events.

The proliferation of mobile devices capable of network communication provides the opportunity to investigate innovative methods of timetable management and communications. The inclusion of this technology would be a key factor in the relationship between scheduling and surveying. As the Space Management Group [4] explains, 'there is a balance between minimising cost and meeting the pedagogical and research needs of staff and the learning and support needs of students.'

Therefore, as we approach 2020 and the carbon agenda, the recommendation is that utilisation analysis needs to be balanced alongside academic delivery, not in place of it. In order to achieve this, the current utilisation survey needs to evolve alongside scheduling.

References

1. MCCOLLUM, B., 1997. The Implementation of a Central Timetabling System in a Large British Civic University. *In:* Lecture Notes In Computer Science, 1408. *Selected papers from the Second International Conference on Practice and Theory of Automated Timetabling II, 1997*. London: Springer-Verlag, pp. 237-253
2. HEFCE, January 2010/01 (updated September 2010), Carbon Reduction target and strategy for higher education in England
3. NATIONAL AUDIT OFFICE, 28 June 1996, The Management of Space in Higher Education Intuitions in Wales
4. SPACE MANAGEMENT GROUP, 2006. Space Utilisation: Practice, performance and guidelines. *SMG* [online], (September 2006). Available at http://www.smg.ac.uk/documents/utilisation.pdf [Accessed 21st January 2013].
5. PRYZIBILLA, J., Timetable Benchmarks for Tertiary Institutions. *Mosaic Space* [online], (February 2010). Available at http://www.mosaicsd.com/Timetabling%20Benchmarks%202010.pdf

MISTA 2013

A heuristic approach to the prisoner transportation problem

Jan Christiaens[1] · **Tony Wauters**[1,2] · **An De Wispelaere**[3] · **Greet Vanden Berghe**[1,2]

1 Introduction

The present paper introduces the Prisoner Transportation Problem (PTP) where prisoners have to be transported at a minimal cost. The PTP is a special case of the vehicle routing problem with pickup and delivery and time windows (PDPTW) (Dumas et al, 1991) The problem has been mentioned by J.G. Partyka (2000). As far as we know, no elaborated case has been published in the literature as of yet. The problem originates from a real-world prisoner transportation case in the Netherlands, where the transportation service has about 300 employees and its head quarter receives 750 transportation requests per day on average. The Prisoner Transportation Problem presents many interesting challenges related to vehicle types, personnel qualifications, service time constraints, exclusionary constraints, etc. A problem formulation and a heuristic approach based on specialized data structures will be discussed presently.

2 The prisoner transportation problem

The PTP can be described as follows. n prisoners have to be transported with a vehicle v from the set of available vehicles V. A prisoner i is held at location l_i^x and has to visit location l_i^y (e.g. a court house or a hospital) from the set of all locations L ($l_i^x, l_i^y \in L$). Driving times and distances between all pairs of locations are given. A visit consists of an outgoing trip, a residence time r_i and an optional return trip. Both trips of one prisoner can be either strongly coupled, e.g. in such a way that the vehicle has to wait and execute the return trip as well, or loosely coupled, e.g. when the return trip can be executed by an other vehicle. On the outgoing trip, a prisoner must be picked up at location l_i^x (e.g. a prison) in time window T_i^a, and delivered at location l_i^y in time window T_i^b. The return trip is similar, i.e. a prisoner must be picked up at location l_i^y in time window T_i^c, provided that the residence time has passed and be delivered at location l_i^x in time window T_i^d. A prisoner requires a service of duration s_i at each

[1]CODeS, KAHO Sint-Lieven
[2]CODeS, KU Leuven KULAK
[3]CONUNDRA, Guldensporenpark 120, B-9820 Merelbeke, Belgium
E-mail: jan.christiaens@kahosl.be

pickup and delivery location. The vehicle is required to wait during the service time. Some additional constraints and objectives have to be considered, including:

Heterogeneous fleet

Different types of vehicles are used. Each vehicle v pertains to a certain vehicle type $q \in Q$ with c_q compartments. Each compartment r of vehicle type q has z_{qr} seats.

Multi-depot

The heterogeneous fleet is distributed over multiple depots $d \in L$. Each depot d has a fleet of vehicles $V_d \subseteq V$. If a vehicle from d is part of a solution, the corresponding route has to start and end at this depot.

Vehicle type and exclusionary constraints

A prisoner can only be transported by a vehicle from a set of allowed vehicle types $Q_i \subseteq Q$. The number of prisoners which are present in a vehicle/compartment at the same time is restricted by constraints. These exclusionary constraints are respectively denoted as vs_{ij} and cs_{ij} for two prisoners i and j.

Simultaneous service times

For each pickup/delivery service, a certain service time is required for embarking/disembarking the prisoner from the vehicle. Services for multiple prisoners at the same location can be executed simultaneously. It is a complex constraint which offers plenty of room for improving solutions obtained by manual planners.

Time limits

The total makespan (driving time + service + waiting time) of a route is limited (e.g. to 540 minutes). Furthermore, the additional time et_{xyi} for transporting prisoner i from location x to y, compared to the direct travel time dt_{xy}, is also limited.

Objectives

The overall objective is to minimize the total cost, which aggregates vehicle costs and personnel costs. The number of required personnel should be minimized, as well as the number of required vehicles and the total travel distance (fuel costs).

Additional personnel scheduling constraints, such as personnel availability and personnel skills, have to be satisfied in any solution.

3 Algorithm and data structures

The high number of constraints makes it hard to generate feasible solutions. Standard moves like 2opt within one route or swaps concerning different routes tend to render solutions to the PTP infeasible. A constructive heuristic, namely a 'best fit', has been developed in combination with an efficient data structure.

The constructive heuristic generates an initial solution based on the best fit implementation. Given a certain visit (thus a pickup and delivery pair), the best fit will return an existing route together with positions in the route scheme such that a pickup and delivery service can feasibly be inserted at minimal additional cost. If no such route has been found, the transportation will be executed in a separate route.

In order to examine every possible insert move, the best fit uses an efficient data structure to speed up the process. This data structure keeps track of precomputed feasibility windows where service times can be executed ensuring that all other services in the route can be executed within the given time windows.

The improvement heuristic operates in a ruin-and-recreate manner as follows. One or more routes are removed from the solution and the visits are redistributed over the

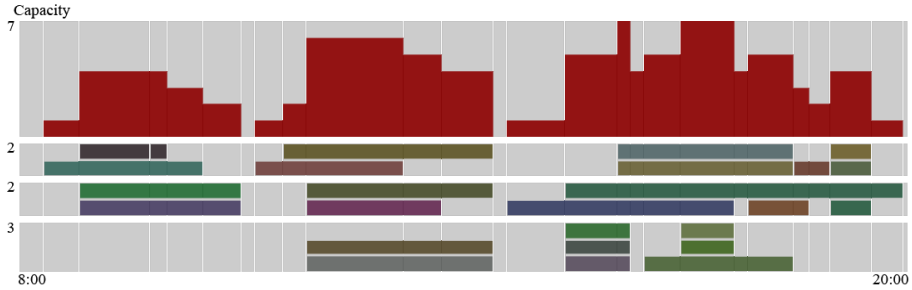


Fig. 1 Prisoner allocation evolving over time for a vehicle with 3 compartments and a total capacity of 7 seats (2+2+3).

existing routes with the same best fit heuristic. If the heuristic cannot find any solution with a lower cost during a certain number of iterations, the process is repeated with a new initial solution.

Besides the time window constraints, a feasible solution needs a feasible prisoner allocation in every vehicle. Figure 1 denotes the occupation of a vehicle with 3 compartments and 7 seats during a particular day. Such an allocation is produced by a separate heuristic, which is called by the best fit each time the feasibility of an insert move is checked. This prisoner allocation heuristic, which is inspired by a well performing graph coloring heuristic (Al-Omari and Sabri, 2006), is shown to produce optimal solutions in more than 95% on a representative dataset. All the data required by this heuristic is integrated in the data structure and maintained during the construction of a route.

4 Datasets and experimental setup

The approach presented in section 3 has been compared with the current planning solutions on a day to day basis. Cost reductions up to 25% have been realized. Less vehicles, less driving distance and less personnel are required to fulfill all transport requests. Furthermore the AS-IS solutions happen to violate particular constraints (e.g. exceeding route makespan, service times executed outside their time window...), while the solutions produced by our approach do not.

5 Conclusion and future work

A new problem has been proposed, i.e. the prisoner transportation problem (PTP). This real-world problem which has many challenging constraints, was successfully solved with an efficient implementation of a best fit heuristic.

We will mimic the actual operations better by including a dynamic aspect, where not all the transport requests are known in advance. Finally, we want to test different scenarios and relaxations of the problem, in order to find out which constraints make the problem very difficult and have the highest influence on the cost function.

Acknowledgements We would like to thank the companies Möbius and Conundra for giving us the opportunity to work on this interesting problem and for delivering the necessary accompanying data.

References

Al-Omari H, Sabri KE (2006) New graph coloring algorithms. American Journal of Mathematics and Statistics 2 (4):739–741

Dumas Y, Desrosiers J, Soumis F (1991) The pickup and delivery problem with time windows. European Journal of Operational Research 54(1):7 – 22

JG Partyka RH (2000) On the road to service. ORMS Today 27(4):26 – 30

MISTA 2013

Two extensions of Lawler's min-max cost algorithm

Nadia Brauner · Gerd Finke · Yakov Shafransky · Dzmitry Sledneu

1 Introduction

We first present, as a reminder, the well-known min-max cost algorithm by Lawler [2] . It applies to surprisingly general functions; its optimality is very easy to prove by a simple job exchange step - and we are just celebrating its 40th anniversary.

Each of n simultaneously available jobs is to be processed without interruption on a single machine, which can process at most one job at a time. Job j, $j = 1, \ldots, n$, has a processing time $p_j \geq 0$. To each of the jobs j there is assigned a nondecreasing cost function $\Phi_j(t)$ that specifies the cost incurred if job j is completed at time t. There are precedence constraints represented by an acyclic directed graph $G = (V,\ E)$. The objective is to find a feasible schedule that minimizes the maximum cost Φ_{max}.

A feasible schedule may be presented by the sequence $\pi = (\pi(1), \ldots, \pi(n))$ of jobs. Here $\pi(k)$ is the job in position k of permutation π.

The corresponding problem $1|prec|\Phi_{max}$ is solved by Lawler's min-max cost algorithm that may be described as follows:

Algorithm 1

1. *Set $k = n$ and $t = \sum_{j=1}^{n} p_j$.*
2. *Let Ω be the set of terminal vertices of graph G,* i.e.*, vertices without successors.*
3. *Find vertex $u \in \Omega$ such that $\Phi_u(t) = \min\{\Phi_v(t) | v \in \Omega\}$ (break ties arbitrarily).*
4. *Set $\pi(k) = u, k = k - 1, t = t - p_u$ and modify graph G by setting $V = V \setminus \{u\}$.*
5. *If $k > 0$, then go to step 2. If $k = 0$, then the optimal permutation is constructed.*

If graph G is represented by its adjacency matrix, then the time complexity of the algorithm is $O(n^2)$. Here we assume that each evaluation of the function $\Phi_j(t)$ can be done in $O(1)$ time.

Nadia Brauner · Gerd Finke
G-SCOP, Université Joseph Fourier
E-mail: Nadia.Brauner@imag.fr, Gerd.Finke@g-scop.inpg.fr

Yakov Shafransky
United Institute of Informatics Problems, NAS of Belarus, Minsk, Belarus
E-mail: shafr-04@yandex.ru

Dzmitry Sledneu
Lund University, Lund, Sweden
E-mail: dzmitry.sledneu@gmail.com

We give two extensions. Lawler's algorithm provides a single very particular optimal solution. The first aim is to give necessary and sufficient conditions of optimality. The second problem is a specialization of Lawler's cost function to a decomposable form, where we add uncertainty.

2 Conditions of optimality

In [3] necessary and sufficient conditions for optimality are given for various single-machine scheduling problems without precedence constraints, including $1||L_{max}$. For our problem $1|prec|\Phi_{max}$ we require all functions $\Phi_j(t)$ to be strictly increasing. Difficulties arise in connection with zero-length jobs, *i.e.*, jobs of negligible processing times.

Let now a feasible permutation π be given. We shall describe a job exchange, as Lawler did in his original paper, but in a slightly modified form. Let us represent the job sequence π schematically as follows:

$$\pi : \boxed{A}\boxed{u}\boxed{B}\boxed{v}\boxed{C}$$

with portions of the sequence A, B and C, where we have singled out a job v and a job u to the left of v with $p_u > 0$, if there is any. We consider only jobs u that are close to v so that the set W of all successor jobs of u in B contains only zero-length jobs. Set $U = (u, W)$ and suppose that the subpermutation U can be moved to the right just following v without violating any precedence constraints, *i.e.* none of the jobs in U is a predecessor of job v. Then we obtain the feasible permutation π':

$$\pi' : \boxed{A}\boxed{B'}\boxed{vU}\boxed{C}$$

where $B' = B \setminus W$. Here it is understood that the inner order of jobs in B' and U is saved from the order in π.

The following definition describes the transitions from π to π' that yield some kind of strict improvement, at least locally.

Definition 1 Given a feasible permutation π and a job v in π with completion time t. We say that job v has a local improvement in π if there exists a non-empty set U, as described above, which satisfies the conditions $\Phi_j(t) < \Phi_v(t)$ for all $j \in U$.

The effect of a local improvement of v is immediate. For π' we get the following properties:

1. The positions of jobs $j \in A \cup C$ are unchanged. Hence these jobs keep the same completion times and costs.
2. Let R be the set of all remaining jobs, then:
   - R occupies in π' the same time interval as in π;
   - the maximum cost of the jobs $j \in R$ is strictly reduced, *i.e.*, $\max_{j\in R} \Phi_j(\pi') < \max_{j\in R} \Phi_j(\pi)$.
3. We have $\Phi_{\max}(\pi') \leq \Phi_{\max}(\pi)$.

Definition 2 Job v is called critical in permutation π if $\Phi_v(\pi) = \Phi_{\max}(\pi)$.

Theorem 1 *If all functions $\Phi_j(t)$ are strictly increasing, then a feasible permutation π is optimal for problem $1\,|\,prec\,|\,\Phi_{\max}$ if and only if there is a critical job in π without any local improvement.*

Notice that Lawler's algorithm constructs an optimal permutation where no job has a local improvement. If the functions $\Phi_j(t)$ are nondecreasing, rather than strictly increasing, the conditions of Theorem 1 are only sufficient for the optimality of π.

3 Decomposable cost functions with uncertainty

Let us specialize the cost function $\Phi_j(t)$ as follows. The cost depends on a nondecreasing function $\varphi(t)$, common for all jobs, and an additional job-specific parameter λ_j. Special cases of such functions include

- lateness : $\Phi_j(t) = t - d_j$, where $d_j \geq 0$ is a due date of job j, and
- delivery times: $\Phi_j(t) = t + q_j$, where $q_j \geq 0$ is the transportation time to send job j to the end user after completion of its processing.

We assume that the actual value of λ_j is unknown, it may take any value from the interval $[\lambda_j^-, \lambda_j^+]$ regardless of the will of a decision maker. The cost function is therefore

$$\Phi_j(t) = \varphi(t) + \lambda_j \text{ where } \lambda_j \in [\lambda_j^-,\, \lambda_j^+]$$

and the corresponding scheduling problem under uncertainty is denoted by

$$1|prec;\, \Phi_j(t) = \varphi(t) + \lambda_j, \lambda_j \in [\lambda_j^-,\, \lambda_j^+]|\Phi_{\max}.$$

Suppose we have given processing times, a function $\varphi(t)$, a graph G and intervals $[\lambda_j^-,\, \lambda_j^+]$ for each $j \in \{1, ..., n\}$. If we choose the values of the additional parameters in any feasible way, *i.e.*, $\lambda_j \in [\lambda_j^-, \lambda_j^+]$ for each $j \in \{1, ..., n\}$, we get a *scenario* I of the problem under uncertainty $1|prec;\, \Phi_j(t) = \varphi(t) + \lambda_j, \lambda_j \in [\lambda_j^-,\, \lambda_j^+]|\Phi_{\max}$ and we denote by $\mathcal{I}$ the set of all scenarios. Clearly, $\mathcal{I}$ is infinite if $\lambda_j^- \neq \lambda_j^+$ for at least one $j \in \{1, ..., n\}$.

Given a scenario $I \in \mathcal{I}$ with parameter values λ_j for each $j \in \{1, ..., n\}$, then we set $\Phi_j(I, t) = \varphi(t) + \lambda_j$ for $j \in \{1, ..., n\}$. For a given permutation π, we write $\Phi_{\max}(I, \pi)$ to indicate the maximal cost of I in π. An optimal permutation for I is called $\pi^*(I)$ and its value is denoted by $\Phi^*_{\max}(I)$.

One of the most popular approaches of dealing with problems under uncertainty is the robust approach [1] aimed to minimize the so-called maximum regret or the maximum deviation of the objective function value from the optimum.

Definition 3 Given a problem $1|prec;\, \Phi_j(t) = \varphi(t) + \lambda_j, \lambda_j \in [\lambda_j^-,\, \lambda_j^+]|\Phi_{\max}$. Define the maximum regret

$$\Delta(\mathcal{I}, \pi) = \max_{I \in \mathcal{I}} \{\Phi_{max}(I, \pi) - \Phi^*_{max}(I)\}.$$

Then a feasible permutation π^0 is called optimal with respect to the maximum regret criterion if it satisfies

$$\Delta(\mathcal{I}, \pi^0) = \min_{\pi \in \Pi} \Delta(\mathcal{I}, \pi)$$

where Π is the set of all feasible permutations.

The only known result for this problem, to our knowledge, is presented by Kasperski in [4], where problem $1|prec|L_{\max}$ is considered under uncertainty of both processing times and due dates. In [4], an $O(n^4)$ algorithm has been developed for constructing a schedule that is optimal with respect to the minmax regret criterion, also in the framework of Lawler's algorithm.

Let us define the subset $\mathcal{H} \subseteq \mathcal{I}$ that contains for each $h \in \{1, \ldots, n\}$ the scenario I_h, which is obtained by setting $\lambda_h = \lambda_h^+$ and $\lambda_j = \lambda_j^-$ for all $j \neq h$. We prove the following identity:

$$\Delta(\mathcal{I}, \pi) = \Delta(\mathcal{H}, \pi)$$

We get within the set $\mathcal{H}$:

$$\Phi_h(I_h, t) = \varphi(t) + \lambda_h^+ \text{ and } \Phi_h(I_j, t) = \varphi(t) + \lambda_h^- \text{ for all } j \neq h.$$

The maximum regret in $\mathcal{H}$ of any job h at time t occurs for scenario I_h, *i.e.*, for all $h \in \{1, ..., n\}$ we get

$$\max_{I_j \in \mathcal{H}} \{\Phi_h(I_j, t) - \Phi^*_{max}(I_j)\} = \varphi(t) + \lambda_h^+ - \Phi^*_{max}(I_h).$$

Thus, we can use Lawlers algorithm to minimize $\Delta(\mathcal{I}, \pi)$.

Algorithm 2

1. *Let a problem* $1|prec; \Phi_j(t) = \varphi(t) + \lambda_j, \lambda_j \in [\lambda_j^-, \lambda_j^+]|\Phi_{\max}$ *be given. For each scenario* $I_h \in \mathcal{H},\ h = 1, \ldots, n$, *calculate* $\Phi^*_{max}(I_h)$.
2. *Apply Algorithm 1 to problem* $1|prec|\Delta_{\max}$ *with the cost functions*

$$\Delta_j(t) = \varphi(t) + \lambda_j^+ - \Phi^*_{max}(I_j)$$

to obtain an optimal permutation π^* *and the minimum value* $\Delta^*_{\max}$.

Theorem 2 *Algorithm 2 solves the minmax regret problem and an optimal solution is given by* $\pi^0 = \pi^*$ *with value* $\Delta(\mathcal{I}, \pi^0) = \Delta^*_{\max}$

It can be shown that the running time of Algorithm 2 is $O(n^2)$ which is the same as the complexity of Lawler's algorithm.

Consider the following interesting special case.

Definition 4 A permutation π is called globally optimal for problem $1|prec; \Phi_j(t) = \varphi(t) + \lambda_j, \lambda_j \in [\lambda_j^-, \lambda_j^+]|\Phi_{\max}$ if π is optimal for all scenarios $I \in \mathcal{I}$.

We get immediately two characterizations for the global optimality.

Theorem 3 *A globally optimal permutation* π *is obtained if Algorithm 2 returns the value* $\Delta^*_{\max} = 0$. *If* $\Delta^*_{\max} > 0$ *then no globally optimal permutation exists. This also means that* π *is globally optimal if, and only if,* π *is optimal for all scenarios* I_h.

Acknowledgements Financial support has been provided in part by the joint BRFFR-PICS project (PICS 5379, BRFFR $\Phi 10\Phi\Pi - 001$).

References

1. Kouvelis P., Yu G. Robust Discrete Optimization and its Applications, Kluwer Academic Publishers, 1997.
2. E.L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. Mangement Sci. 1973, Vol. 19 N 5, P. 544–546.
3. Y. Lin, X. Wang. Necessary and sufficient conditions of optimality for some classical scheduling problems. European Journal of Operational Research. 2007, Vol. 176, P. 809–818.
4. Kasperski A. Minimizing maximal regret in the single machine sequencing problem with maximum lateness criterion, Operations Research Letters, 2005, Vol. 33, P. 431–436.

MISTA 2013

An Evolutionary Hill Climbing Algorithm for Dynamic Optimisation Problems

Ayad Mashaan Turky • Salwani Abdullah • Barry McCollum • Nasser R. Sabar

1 Introduction

In contrast to stationary problems where the problem parameters are static, dynamic optimisation problems (DOPs) present a challenging research area to the community [1]. This is mainly because, changes can occur any time during the optimisation course. Hence, when such a change happens (for example a change in the fitness landscape), the previous local optimal solution may no longer be relevant [2,3]. It is widely known that local search algorithms usually face a serious challenge for dynamic optimisation problems because they operate on a single solution which can easily get stuck in a local optima [2]. In order to overcome this shortage, local search algorithms should be re-started from scratch. Although, it seems a reasonable solution, restarting the algorithm from scratch will be time consuming as it may lose some promising information. A promising trend to solve this issue is to hybridise a local search with a population-based method such as a genetic algorithm [2] [3]. The literature shows that population-based methods have been widely and successfully used to tackle dynamic optimisation problems (DOPs). The work in this paper is motivated by the success of the above mentioned methods. Thus, we propose the evolutionary hill climbing algorithm which combines the population of solutions with the hill climbing local search for dynamic optimisation problems. Our aim here is to use a population of solutions (instead of a single solution) in order to keep track of any changes during the search process, since the population of solutions are scattered over the entire search space and consequently will increase the

Ayad Mashaan Turky
Data Mining and Optimization Research Group (DMO),
Center for Artificial Intelligence Technology, Universiti Kebangsaan Malaysia
E-mail: ayadalrashid@gmail.com

Salwani Abdullah
Data Mining and Optimization Research Group (DMO),
Center for Artificial Intelligence Technology, Universiti Kebangsaan Malaysia
E-mail: salwani@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

Nasser R. Sabar
[1]Data Mining and Optimization Research Group (DMO),
Center for Artificial Intelligence Technology, Universiti Kebangsaan Malaysia,
[2]The University of Nottingham Malaysia Campus
43500 Semenyih Selangor, Malaysia

diversification of the algorithm. A hill climbing algorithm is applied to refine the promising solution generated by the population method. Once the solution cannot be improved within a certain number of iterations, the hill climbing algorithm will restart the search by selecting a new solution from the population (instead of generating a new one).

This idea will not only reduce the computation time, but will also help the search to focus on a promising point, as they already been improved previously and been modified by the population-based method. The performance of the proposed method is validated over the well-known dynamic optimisation functions i.e., OneMax, Plateau, Royal Road and Deceptive. Experimental results show that the proposed algorithm is able to achieve competitive results when compared to other available methods.

The rest of this paper is outlined as follows: Section 2 describes the proposed method. The experimental results and relevant discussion are presented in Section 3. Section 4 concludes the work presented in this paper.

2 Proposed Method

In this section, we present the proposed evolutionary hill climbing algorithm (coded as Pop-HC), which combine the population of solutions with the hill climbing local search for solving dynamic optimization problems, as depicted in Fig. 1. The proposed method consists of two phases. In the first phase, an evolutionary process is used to perturb the generated solutions. The second phase uses a hill climbing algorithm to improve the solution generated by the evolutionary process. These two phases are successively invoked until a stopping condition is satisfied (e.g. number of iterations). The idea behind the use of the population of solutions is to provide initial solutions to the hill climbing algorithm due to the fact that the hill climbing algorithm can easily get stuck in local optima, and re-start the algorithm from scratch will be time consuming as it may lose some promising areas. Hence, by using a population of solutions, we may get a better result than re-start the solution from scratch, because the search process can focuses on the promising point, as the solutions have been improved previously and also been modified by the population-based method.

a. Solution representation, initial population generation method and the neighbourhood structure

In this work, we use a binary representation scheme where each bit in the solution is represented by either ‘1’ or ‘0’ with a size of 100 (see Table 1 for parameter setting). The population of solutions is generated by randomly filling up each bit by either ‘0’ or ‘1’. The solution is directly used in the objective function to calculate the fitness of the generated solution, i.e. calculate the number of ‘1s’. The neighbourhood of the current solution is generated by randomly selecting one bit and changes it to its complement, e.g., ‘1’ to ‘0’ or vice versa.

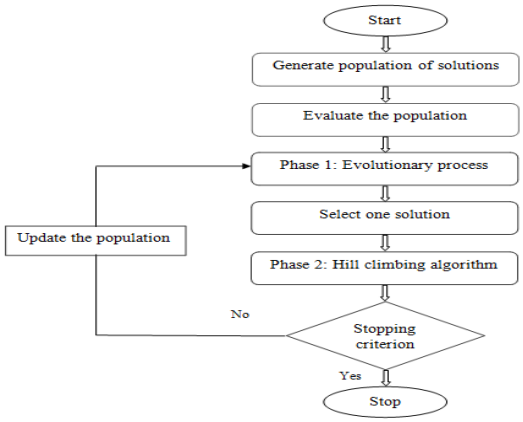


Fig. 1 Proposed approach, Pop-HC

b. Phase 1: Evolutionary process. In this phase, the evolutionary process begins with a population of solutions. Then, we evaluate the population of solutions. The evolutionary process contains a population of solutions as well as a perturbation mechanism. The perturbation mechanism uses a flip-flop operator in order to help the solution to get out from the local optima (mutation operators). Thus, the role of the evolutionary process is to perturb the solution that is already improved by the hill climbing algorithm. In this work, a simple perturbation mechanism is employed to modify one solution (or solutions) that has (have) been improved by the hill climbing algorithm by randomly selecting one bit and change its value to '1' if the current value is '0' and vice versa. This helps the algorithm to jump out from the local optima if the same solution is selected again.

c. Phase 2: Hill climbing algorithm. In this phase, the hill climbing algorithm is invoked. The algorithm starts with a single solution (S) that is randomly selected from the population of solutions. Set S as a best solution (BestSol). The algorithm iteratively generates a neighbor of the solution (S*) with an aim to obtain a better solution than the current one. The neighbor of the solution is generated by selecting one bit at random and change its value to its complement (i.e. change "1" to "0" or "0" to "1"). If the generated solution is better than the current one (S), the current solution is updated (S=S*) and will be used in the next iteration. If the current solution is better than the best solution then the best solution is updated (BestSol=S). This process is repeated for a certain number of iterations which is set by user. Once the stopping condition is satisfied (MaxIter), the solution will be added into the population if its fitness is better than the worst solution in the population.

3 Results and Discussions

In this section, we evaluate the proposed method on the well-known dynamic optimisation test functions. The tested functions are: OneMax, Plateau, Royal Road and Deceptive [1]. These functions (which use a binary-encoding) have been widely used by many researchers. Originally, they are stationary functions and have been changed to be dynamic test functions. In order to generate dynamic environments from these binary-encoded stationary test functions, we use the dynamic generator that was proposed in [4,5]. To use this dynamic generator, two parameters need to be set in advance which are the speed of change, *T* (which represents the periodically of changes) and *P* (which represents the amount of change). These two parameters are set based on the work reported in [1]. The parameter setting for the proposed method is listed in Table 1.

Table 1: Parameters settings

Parameter	**Value**
Population Size	50
Solution size	100
Number of iterations	100
T	100
P	0.9

The proposed method is compared against the well-known methods in the literature. The methods in comparison are:

- MIGA: Genetic algorithm with memory based immigrants [6].
- MEGA: memory-enhanced Genetic algorithm [7].
- AHMA: Memetic algorithm with the AHC operator [1].
- MRIGA: Genetic algorithm with memory and random immigrants schemes[6]

The average results of Pop-HC over 30 runs (with different initial solutions and seeds) as well as other methods in comparison are presented in Table 2. The presented results indicate that

Pop-HC outperforms other methods on the Plateau test function and competitive results on OneMax, Royal Road and Deceptive. Pop-HC is at the 2nd, 3rd and 4th rank on OneMax, Royal Road and Deceptive functions, respectively. The results also show that the percentage of deviation from the best result is quite competitive. Individual comparison shows that the Pop-HC is able to produce better results than MIGA, MEGA, AHMA and MRIGA in between 1 and 2 instances, and can be considered as competitive results across all tested functions. Note that, the methods in comparison here did not attempt on all test functions. We believe this is due to the use of the population of solutions with the hill climbing algorithm in order to preserve the diversity during the search process. The results also indicate that a mechanism to improve the diversification might lead to better results. Thus, in the future work we intend to integrate other diversification methods and tested their performance on the same instances.

Table 2: Results Comparison

Function name	Pop-HC	% Deviation	MIGA	MEGA	AHMA	MRIGA
OneMax	86.35	9.94	94.0	79.3	**95.89**	80.8
Pleateau	**74.21**	**	-	-	62.88	-
Royal Road	51.11	2.68	-	-	**52.52**	-
Deceptive	72.56	15.38	71.1	83.1	**85.75**	68.6

"-": no results reported. "**": our algorithm better than others.

.

4 Conclusions

The evolutionary hill climbing algorithm (that hybridised the population-based method with the hill climbing algorithm) for dynamic optimisation problems has been presented in this paper. The role of the population-based method is to generate a population of solutions to be used as an initial solution for the hill climbing algorithm. It also maintains the diversity of solution by using a simple perturbation mechanism in modifying the solution. Whilst, the hill climbing algorithm aims to further improve the randomly selected solution from the population. The performance of the proposed algorithm is assessed on the dynamic optimisation functions and experimental results show that the proposed method obtained competitive results when compared to the methods in the literature.

References

1. Wang H, Wang D, Yang S (2009) A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems. Soft Computing - A Fusion of Foundations, Methodologies and Applications 13 (8):763-780. doi:10.1007/s00500-008-0347-3
2. Talbi EG (2002) A Taxonomy of Hybrid Metaheuristics. Journal of Heuristics 8 (5):541-564. doi:10.1023/a:1016540724870
3. Yang S, Ong YS, Jin Y (2007) Evolutionary computation in dynamic and uncertain environments, vol 51. Springer-Verlag, Berlin Heidelberg.,
4. Shengxiang Y Non-stationary problem optimization using the primal-dual genetic algorithm. In: Evolutionary Computation, CEC. The Congress on, 8-12 Dec. 2003 2003. pp 2246-2253 Vol.2243
5. Yang S, Yao X (2005) Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Computing - A Fusion of Foundations, Methodologies and Applications 9 (11):815-834. doi:10.1007/s00500-004-0422-3
6. Yang S (2008) Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. Evolutionary Computation 16 (3):385-416
7. Yang S Memory-based immigrants for genetic algorithms in dynamic environments. In: Proc. of the Genetic and Evol. Comput. Conf., vol. 2, pp,GECCO, 2005. ACM, pp 1115-1122

MISTA 2013

Hierarchical Control Model for Several Stochastic Network Projects

Gonik Aharon

Abstract
Almost every company is handling simultaneously several projects. The common practice is to supervise each project independently, neglecting levelling company Cash-Flow, optimization of limited Resources and supporting projects that are "Legging-behind". A hierarchical control model is suggested which at any dynamic control point determines: 1.Optimal budget values reassigned from the company to each project. 2. Optimal resource delivery schedules from a central control source for all the projects activities. In order to - maximize the probability of meeting the deadline of the slowest project. A Critical-Path, flowing across all the projects (as a unified system), which replaces the Critical- Paths of the particular projects, will be established via the On-Line simulated control.

MISTA 2013 Challenge: Finalists

The papers in this section are those that were submitted as part of the MISTA 2013 challenge. These contributions were subject to the same peer review processes as the other papers/abstracts in these proceedings.

The subject of the MISTA 2013 challenge was the multi-mode resource-constrained multi-project scheduling problem (MRCMPSP). Multiple projects have to be scheduled simultaneously taking into account the availability of local and global resources. The MRCMPSP has a large practical relevance, including examples in the construction and production sector.

MISTA 2013

A hybrid local search algorithm integrating an exact CP-SAT approach to solve the multi-mode resource-constrained multi-project scheduling problem

Alexander Schnell

1 Introduction

The multi-mode resource-constrained project scheduling problem (MRCPSP) is a generalization of the single-mode RCPSP (SRCPSP) where an additional mode-assignment step has to be considered. The aim is to find the best mode-assignment for a number of jobs subject to non-renewable resource constraints such that the optimal schedule for the resulting SRCPSP (if existing) optimizes a specific objective function. Note, that for the MRCPSP finding a feasible mode-assignment subject to more than 1 non-renewable resource constraint is NP-complete [6].

For the SRCPSP recent exact algorithms combine Branch and Bound-based optimization search with principles from Constraint Programming (CP) and Boolean Satisfiability Solving (SAT) (See [3], [5] and [9]). The most successful and currently best exact algorithm for the SRCPSP is the one presented in [9]. In a preceding project, we aimed at generalizing the above principles to the MRCPSP. Therefore, we implemented a new constraint handler `cumulativemm(...)` for the Constraint Integer Programming solver SCIP [1] which extends the existing `cumulative(...)`-constraint for the SRCPSP to the MRCPSP. In concrete, we generalized the propagation algorithms and the explanations [10] for the Conflict Analysis (CA)-Plugins of SCIP by also considering the mode assignment step[1].

For the MISTA competition [8], we implemented a model with SCIP which applies our constraint handler `cumulativemm(...)` for the modeling and solution of the multi-mode resource-constrained multi-project scheduling problem (MRCMPSP) (See Sect. 2). The advantages compared to the solution of a binary integer programming (BIP) formulation [7] is the application of specialized propagation algorithms and SAT techniques which have turned out to be very efficient for the SRCPSP [9].

We are able to solve the competition instances with at most 2 projects to optimality by the solution of the model from Sect. 2 in less than 15s. For the remaining instances we implemented a local search algorithm (See Sect. 3) whereas the construction and

Alexander Schnell
University of Vienna
E-mail: alexander.schnell@univie.ac.at

[1] We plan to publish the approach and the results in an extra paper.

improvement step are based on the solution of variations of the model of Sect. 2. The results obtained by the exact and heuristic algorithm can be found in Sect. 4.

2 Exact Solution

When using our `cumulativemm`(...)-constraint and the notation provided by the MISTA competition organizers [8], the MRCMPSP can be formulated as follows with SCIP:

$$\min \quad H_{\max} \cdot \left(\sum_{i \in P} s_{i,|J_i|+1,1} - \sum_{i \in P} (r_i + CPD_i) \right) + s_g \tag{1}$$

$$s.t. \quad \sum_{k \in M_{i,j}} x_{i,j,k} = 1,$$

$$\forall i \in \mathcal{P},\ \forall j \in J_i \cup \{0, |J_i|+1\} \tag{2}$$

$$s_{i,j,k} + d_{i,j,k} \cdot x_{i,j,k} \leq s_{i,j',k'},$$

$$\forall i \in \mathcal{P},\ j \prec j',\ k \in M_{i,j},\ k' \in M_{i,j'} \tag{3}$$

$$s_{i,|J_i|+1,1} \leq s_g, \qquad \forall i \in \mathcal{P} \tag{4}$$

$$\sum_{j \in J_i} \sum_{k \in M_{i,j}} r^{\nu}_{i,j,k,l} \cdot x_{i,j,k} \leq c_{i,l}, \qquad \forall i \in \mathcal{P},\ l \in L^{\nu}_i \tag{5}$$

$$\texttt{cumulativemm}(\hat{\mathbf{s}}^i, \hat{\mathbf{x}}^i, \hat{\mathbf{d}}^i, \hat{\mathbf{r}}^i_l, c_{i,l}), \qquad \forall i \in \mathcal{P}, l \in L^{\rho}_i \tag{6}$$

$$\texttt{cumulativemm}(\hat{\mathbf{s}}, \hat{\mathbf{x}}, \hat{\mathbf{d}}, \hat{\mathbf{r}}_g, c_g), \qquad \forall g \in G^{\rho} \tag{7}$$

$$s_{i,j,k} \in \left\{ lb(s_{i,j,k}), \ldots, ub(s_{i,j,k}) \right\},$$

$$x_{i,j,k} \in \{0,1\},$$

$$\forall i \in \mathcal{P},\ j \in J_i \cup \{0, |J_i|+1\},\ k \in M_{i,j} \tag{8}$$

where

$$\hat{\mathbf{s}}^i = \mathbf{s}^{i,0} \circ \cdots \circ \mathbf{s}^{i,|J_i|+1}, \qquad \text{where } \mathbf{s}^{i,j}_k = s_{i,j,k},$$
$$\forall\, i \in \mathcal{P},\ j \in J_i \cup \{0, |J_i|+1\}, k \in M_{i,j} \tag{9}$$

$$\hat{\mathbf{x}}^i = \mathbf{x}^{i,0} \circ \cdots \circ \mathbf{x}^{i,|J_i|+1}, \qquad \text{where } \mathbf{x}^{i,j}_k = x_{i,j,k},$$
$$\forall\, i \in \mathcal{P},\ j \in J_i \cup \{0, |J_i|+1\}, k \in M_{i,j} \tag{10}$$

$$\hat{\mathbf{d}}^i = \mathbf{d}^{i,0} \circ \cdots \circ \mathbf{d}^{i,|J_i|+1}, \qquad \text{where } \mathbf{d}^{i,j}_k = d_{i,j,k},$$
$$\forall\, i \in \mathcal{P},\ j \in J_i \cup \{0, |J_i|+1\}, k \in M_{i,j} \tag{11}$$

$$\hat{\mathbf{r}}^i_l = \mathbf{r}^{i,0,l} \circ \cdots \circ \mathbf{r}^{i,|J_i|+1,l}, \qquad \text{where } \mathbf{r}^{i,j,l}_k = r^{\rho}_{i,j,k,l},$$
$$\forall\, i \in \mathcal{P},\ j \in J_i \cup \{0, |J_i|+1\}, k \in M_{i,j}, l \in L^{\rho}_i \cup G^{\rho} \tag{12}$$

and

$$\hat{\mathbf{s}} = \hat{\mathbf{s}}^0 \circ \cdots \circ \hat{\mathbf{s}}^{n-1} \tag{13}$$

$$\hat{\mathbf{x}} = \hat{\mathbf{x}}^0 \circ \cdots \circ \hat{\mathbf{x}}^{n-1} \tag{14}$$

$$\hat{\mathbf{d}} = \hat{\mathbf{d}}^0 \circ \cdots \circ \hat{\mathbf{d}}^{n-1} \tag{15}$$

$$\hat{\mathbf{r}}_l = \hat{\mathbf{r}}^0_l \circ \cdots \circ \hat{\mathbf{r}}^{n-1}_l \tag{16}$$

$$\hat{\mathbf{r}}_g = \hat{\mathbf{r}}^0_g \circ \cdots \circ \hat{\mathbf{r}}^{n-1}_g \tag{17}$$

Note that the variables $s_{i,j,k}$ represent the starting times of the activity $j \in J$ of project $i \in \mathcal{P}$ in mode $k \in M_{i,j}$ and the variables $x_{i,j,k}$ model the processing of the activity $j \in J$ of project $i \in \mathcal{P}$ in mode $k \in M_{i,j}$. Hereby, $s_{i,|J_i|+1,1}$ are the starting time variables for the dummy jobs representing the ending of project $i \in \mathcal{P}$ with $M_{i,|J_i|+1} = \{1\}$.

The objective function (1) consists of a combination of the total project delay (TPD) and the total makespan (TMS) s_g, which is evaluated through (4). Thereby, we multiply the TPD with a sufficiently big makespan H_{max} of the complete multi-project instance to guarantee the lexicographical optimization of the two values.
(3) are multi-mode precedence constraints. With (5), we assure that the non-renewable resource limits are not exceeded.

In (6) and (7), we apply our `cumulativemm`(...)-constraint to model the resource constraints for the local and global renewable resources. Thereby, we use the variable vectors $\hat{\mathbf{s}}^i$, $\hat{\mathbf{s}}$, $\hat{\mathbf{x}}^i$ and $\hat{\mathbf{x}}$. In this context, the operator $\circ$ is defined as the concatenation of two vectors, whereas the vector $\mathbf{c} = \mathbf{a} \circ \mathbf{b}$ is obtained by appending the elements of $\mathbf{b}$ coordinate-wise to $\mathbf{a}$.

3 Local Search for the MRCMPSP

Our local search algorithm consists of a construction and an improvement step. In the construction step, we firstly try to solve every single project separately with SCIP with the aim of makespan minimization. After a certain time limit L, we stop the solution process and store the currently best MRCPSP solutions. Then, we try to find the best sequence according to which the projects are ordered. For this we solve a one-machine problem with given release dates and possible preemption with the aim of minimizing the complete project delay (equivalent to the first term in (1)). From the optimal sequence and the starting times of the solutions of every single MRCPSP project, we generate a feasible schedule for the MRCMPSP. Based on this schedule, we generate an activity list, which sorts all MRCMPSP activities in ascending order of their starting times. After applying the serial schedule generation scheme (SSGS) [4, p.144 ff.] to the latter activity list, we obtain a feasible starting solution for our local search algorithm.

Our improvement step consists of the search for a new solution in one of the four following neighborhoods. Two neighborhoods are based on the relaxation of at most 2 projects in the currently best schedule. For the first one we randomly relax 1-2 projects and for the second one we relax 1-2 projects for which the relaxation would lead to the smallest lower bound on the objective function (1).
The third neighborhood is based on the relaxation of a random number of activities which are scheduled at the end in the incumbent solution.

For the fourth neighborhood we only relax the mode variables in the incumbent solution. With this neighborhood we try to find a mode-assignment which minimizes the complete renewable resource energy [2, p.15 ff.]. After having found a new mode-assignment, we apply the SSGS to the incumbent solution with the new mode-assignment to obtain a new feasible schedule.
The search for a new schedule in the neighborhoods is based on solving a variation of the model presented in Sect. 2 with SCIP within a given time limit. Note, that we store information about the models already solved, so that we do not solve the same model twice.

If a solution in one of the neighborhoods has a better objective value than the incumbent solution, we update the best solution. Note that the neighborhoods are randomly chosen, whereas we assign a higher selection probability to the first two neighborhoods.

4 Results

Our exact and heuristic approach was tested on the Vienna Scientific Cluster. Thereby, the cluster nodes integrate a X86-64 architecture running under GNU/Linux with Intel X5550 processors of 2,66GHz and with 24GB RAM. We implemented our approaches in `C/C++` in combination with SCIP 3.0.0. The Tables 1 and 2 show the results for the instance sets `A` and `B` [8]. One can see the average TPD and TMS, the best TPD and TMS (belonging to the best TPD) based on 10 runs for every instance. Moreover, the tables contain the total solution times (t_{tot}) and the gaps to the best known solutions (G_{BK}) for the instance set `A` [8]. Note, that for the instances `A-1-A-3`, we found the optimal solution in less than 15s by solving the model of Sect. 2 with SCIP. For the remaining instances, we applied the heuristic algorithm of Sect. 3 with the time limit $t_{tot} = 300s$ given by the MISTA competition organizers [8].

	avg. TPD	avg. TMS	best TPD	best TMS	t_{tot} (s)	G_{BK} (%)
`A-1`	1.00	23.00	1	23	0.67	0.00
`A-2`	2.00	41.00	2	41	14.35	0.00
`A-3`	0.00	50.00	0	50	4.83	0.00
`A-4`	74.73	54.82	65	45	300	0.00
`A-5`	208.11	123.44	192	121	300	25.49
`A-6`	273.90	132.00	221	114	300	50.34
`A-7`	689.60	208.30	676	204	300	13.42
`A-8`	433.60	172.80	392	168	300	29.80
`A-9`	404.20	159.40	370	158	300	65.92
`A-10`	1313.50	369.80	1286	368	300	32.71
						21.77

Table 1: Results for the instance set `A`

	avg. TPD	avg. TMS	best TPD	best TMS	t_{tot} (s)
`B-1`	455.60	145.50	441	142	300
`B-2`	549.10	173.60	520	170	300
`B-3`	743.80	229.70	727	222	300
`B-4`	1832.10	328.40	1689	309	300
`B-5`	1321.90	303.30	1203	289	300
`B-6`	1290.80	265.80	1222	252	300
`B-7`	1418.40	297.10	1321	291	300
`B-8`	3848.90	599.80	3713	604	300
`B-9`	7600.20	1059.30	6993	1022	300
`B-10`	3634.70	468.10	3422	466	300

Table 2: Results for the instance set `B`

5 Conclusion

In a preceding project, we have implemented a new constraint handler `cumulativemm(...)` for the solver SCIP to tackle RCPSP instances with multi-mode jobs. For the MISTA competition 2013 [8], we applied this constraint to exactly and heuristically solve the MRCMPSP. We are able to solve the competition instances with at most 2 projects to optimality. For the remaining instances, we implemented a local search algorithm which relaxes parts of the complete problem and tries to solve the remaining problem to optimality with SCIP. With an increasing number of projects, even the exact solution of the relaxed problems can take a large amount of time. Thus, we assume a potential for an ample improvement of the results by relaxing greater parts of the incumbent solutions and optimizing the relaxed problems by metaheuristics like e.g. genetic algorithms.

References

1. Achterberg, T.: SCIP: solving constraint integer programs. Mathematical Programming Computation **1**(1), 1–41 (2009)
2. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based scheduling: applying constraint programming to scheduling problems. Springer, Netherlands (2001)
3. Berthold, T., Heinz, S., Lübbecke, M., Möhring, R., Schulz, J.: A constraint integer programming approach for resource-constrained project scheduling. In: A. Lodi, M. Milano, P. Toth (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, *Lecture Notes in Computer Science*, vol. 6140, pp. 313–317. Springer Berlin Heidelberg (2010)
4. Brucker, P., Knust, S.: Complex scheduling. Springer Berlin, Germany (2006)
5. Horbach, A.: A boolean satisfiability approach to the resource-constrained project scheduling problem. Annals of Operations Research **181**(1), 89–107 (2010)
6. Kolisch, R., Drexl, A.: Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE transactions **29**(11), 987–999 (1997)
7. Kolisch, R., Sprecher, A.: PSPLIB-a project scheduling problem library. European Journal of Operational Research **96**(1), 205–216 (1997)
8. MISTA 2013: Challenge: the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem. `http://allserv.kahosl.be/mista2013challenge/index.html` (2013)
9. Schutt, A., Feydy, T., Stuckey, P.: Explaining time-table-edge-finding propagation for the cumulative resource constraint. Arxiv preprint arXiv:1208.3015 (2012)
10. Schutt, A., Feydy, T., Stuckey, P., Wallace, M.: Explaining the cumulative propagator. Constraints **16**(3), 250–282 (2011)

MISTA 2013

Iterated Variable Neighborhood Search for the resource constrained multi-mode multi-project scheduling problem

Some comments on our contribution to the *MISTA 2013 Challenge*

Martin Josef Geiger

1 Introduction

The resource constrained multi-mode multi-project scheduling problem (RCMMMPSP) is a notoriously difficult combinatorial optimization problem. For a given set of activities, feasible execution mode assignments and execution starting times must be found such that some optimization function, e. g. the makespan, is optimized. When determining an optimal (or at least feasible) assignment of decision variable values, a set of side constraints, such as resource availabilities, precedence constraints, etc., has to be respected.

In 2013, the *MISTA 2013 Challenge* stipulated research in the RCMMMPSP. It's goal was the solution of a given set of instances under running time restrictions. We have contributed to this challenge with the here presented approach.

2 Problem description and implications

In the RCMMMPSP of the *MISTA 2013 Challenge*, several small projects, each of which comprises a set of activities, have to be integrated into a single overall plan. A release date of each project has to be respected, and resources that are jointly used by several projects may exist. Two types of resources exists: renewable and non-renewable. The latter are always specific to each project, i. e. they are never shared among the projects.

The quality of the obtained schedule is evaluated by two objective functions, namely the minimization of the total project delay (TPD), and the total makespan (TMS), which are considered in a lexicographical ordering with TPD over TMS.

For a more detailed description of the RCMMMPSP, we refer to `http://allserv.kahosl.be/mista2013challenge/`.

Martin Josef Geiger
Helmut-Schmidt-University, University of the Federal Armed Forces Hamburg
E-mail: m.j.geiger@hsu-hh.de

2.1 A remark on the multi-project characteristic

It is possible to combine the set of projects into an overall super-project by relabeling the numbering of the activities. Quite intuitively, this polynomial-time transformation allows for a rather straight-forward representation of alternatives, in which each activity is represented by a unique index $i, i = 1, \ldots, n$ (see section 3.1 below). However, we cannot rule out the possibility that a separate treatment/ optimization of the different small sub-projects is not of any potential value.

2.2 A note on the mode assignment

In the general case, i. e. the case of at least two non-renewable resources, the determination of a feasible mode assignment is $\mathcal{NP}$-complete [1]. This means that, depending on the datasets, this sub-aspect of the problem may pose a problem on it's own. Any heuristic approach for selecting mode assignments is therefore risky, especially under running time restrictions.

2.3 Two remarks on the objective functions

It can be shown that both objectives are *regular* functions. It is therefore possible to restrict the computations to active plans only, without excluding the optimum. This is an important observation, that allows the use of a serial scheduling scheme (SSS) [2,3]. In such a scheme, a schedule is constructed by sequentially assigning activities into a schedule, one activity at a time. An earliest-possible-('left-shifted-')scheduling-rule may here be employed for obtaining an active schedule. This implies that an optimization approach can then search the sequence in which the activities are placed in the final schedule, instead of searching the activities' starting times themselves.

Note that, in the general case, a combination of the two criteria into a single evaluation function is not possible. If however we may safely assume that, for a particular set of problem instances (datasets), TMS $< \alpha$, then a combination of TPD and TMS is possible such that: $F = \alpha$ TPD + TMS. In the context of the *MISTA 2013 Challenge*, the organizers guarantee a value of $\alpha = 100{,}000$.

3 Solution approach

3.1 Representation of alternatives

An alternative $\mathbf{X}$ is associated with two vectors $\mathbf{M} = (m_1, \ldots, m_n)$ and $\mathbf{S} = (s_1, \ldots, s_n)$. $\mathbf{M}$ encodes for each activity i the choice of the execution mode m_i, and $\mathbf{S}$ is a permutation of the activity indexes. In the following, let $\mathcal{X}$ denote the set of feasible alternatives.

While the information in $\mathbf{M}$ is obvious, $\mathbf{S}$ can be decoded into a schedule by a SSS. In this transformation, the feasibility of $\mathbf{X}$ with respect to $\mathbf{S}$ is maintained at all times as we adhere to the precedence constraints of activities while carrying out the procedure. Scheduling conflicts are solved such that priority is given in accordance with the sequence of activities in $\mathbf{S}$. Using such a SSS implies that any permutation of activity indexes leads to a feasible schedule.

3.2 Generation of first feasible alternatives

The generation of initial alternatives is twofold:

1. First, a random mode assignment $\mathbf{M}$ is computed. If $\mathbf{M}$ is infeasible, i. e. by exceeding the capacity of the non-renewable resources, a repair procedure is triggered, which randomly alters values m_i in $\mathbf{M}$ until the feasibility of the mode choices is restored. After a maximum number of unsuccessful repair attempts, $\mathbf{M}$ is randomly reconstructed from scratch. Note that the repair procedure is guided towards a feasible mode assignment by minimizing the overall excessive use of the non-renewable resources (this assumes a value of 0 for a feasible $\mathbf{M}$).
   In the light of section 2.2, such a procedure could turn out to be problematic, as it does not guarantee the identification of the feasible mode assignment. It has therefore been tested on all 1,642 resource constrained multi-mode project scheduling datasets which were delivered in the *MISTA 2013 Challenge*. In conclusion, the procedure repeatedly and consistently managed to create feasible mode vectors for all instances.
2. Second, a scheduling sequence $\mathbf{S}$ is constructed. The SSS described above is used here. For the first alternative, scheduling conflicts are solved by giving priority to activities with smaller lower bounds on their possible starting times.

3.3 Neighborhood search

The initially constructed alternative $\mathbf{X}$ is then improved by means of local search. The proposed concept is based on the ideas of Variable Neighborhood Search [4] (VNS) and Iterated Local Search [5]. Starting from an initial (feasible) alternative, neighboring alternatives are generated by means of several neighborhood operators and tested for acceptance. Once an alternative is reached that cannot be improved by any operator (a local optimum), a (possibly worsening) perturbation move is performed and search continues from this alternative.

Four different neighborhood operators have been implemented:

1. Exchange (EX), which exchanges the position of two activities in $\mathbf{S}$. We denote the set of the neighbors of $\mathbf{X}$ by the operator EX as $\mathrm{EX}(\mathbf{X})$. A neighboring solution $\mathbf{X'} \in \mathrm{EX}(\mathbf{X})$ is accepted iff $F(\mathbf{X'}) < F(\mathbf{X})$.
2. Inversion (INV), which inverts a subsequence of activities within $\mathbf{S}$. We accept $\mathbf{X'} \in \mathrm{INV}(\mathbf{X})$ iff $F(\mathbf{X'}) < F(\mathbf{X})$.
3. Single mode change (SMC), which changes the mode assignment of a single element in $\mathbf{M}$. $\mathbf{X'} \in \mathrm{SMC}(\mathbf{X})$ is accepted iff $F(\mathbf{X'}) < F(\mathbf{X}) \wedge \mathbf{X'} \in \mathcal{X}$.
4. Two modes changes (TMC), which changes the mode assignments of two elements in $\mathbf{M}$. We accept $\mathbf{X'} \in \mathrm{TMC}(\mathbf{X})$ iff $F(\mathbf{X'}) \leq F(\mathbf{X}) \wedge \mathbf{X'} \in \mathcal{X}$. Note that this acceptance rule differs from the ones of EX, INV, and SMC: Accepting alternatives of equal quality appears to have a beneficial effect for this operator, possibly due to some diversification effect that results from it.

In case of smaller instances ($n < 307$: this value has been determined by a series of experiments), each neighborhood is repeatedly searched until no improvement can be identified. Then, the next neighborhood operator is taken. For larger instances, this does not appear to be an appropriate strategy. Here, we search each neighborhood,

and then move directly to the next one, thus iterating through the set of neighborhood operators more quickly.

Escaping local optima: The perturbation move consists in randomly changing the execution mode of a single activity. This move is followed by calling the randomized mode-repair-procedure, which, if required, alters several other mode assignments and thus implicitly provides some sort of diversification.

3.4 A note on the parallelization of the approach

In case of smaller instances ($n < 307$), the above presented local search is simply executed in parallel (one master thread, four worker threads). Once a worker thread reaches a local optimum, the currently best found alternative is updated in the master thread and search continues with this best known alternative (perturbation + VNS).

Larger instances come with the difficulty of obtaining a local optimum in the first place. Therefore, all available CPU-cores are concentrated on a single solution by dividing up the investigation of a neighborhood among them.

4 Experiments and results

4.1 Implementation details and hardware

Our algorithm is coded using Microsoft Visual Studio 2012. The program therefore requires a Microsoft "Windows" operating system and a current version of the ".NET Framework" (version 4.5). We have used an Intel X5550 processor, running at 2.67 GHz for the test runs. Despite the availability of eight cores on our machine, only four threads of the local search algorithm are executed in parallel. The target platform for the implemented code optimizations is 64-bit (OS/CPU).

The benchmark program of the 2011 International Timetabling Competition gave, on the organizers machine, which is an Intel Core i7-2600 (3.4 GHz), an execution time of 619 seconds. On our Intel X5550, the very same program reported a running time of 804 seconds (average of 5 trials). We have concluded that the Intel X5550 should be given 29.9% more computing time, resulting in a permitted computing time of 389 seconds for each run. Note that the comparison of the ITC 2011 benchmark program pretty much reflects the different clock speeds of the CPUs.

4.2 Results and execution of the program

Table 1 reports the results of 20 test runs, i. e. the quality of the best found alternatives and the median values. We can observe a considerable deviation of the best value from the median. This indicates that the approach is not consistently solving the datasets each and every time it is executed. While this is typical for a stochastic solver, especially under, for the larger datasets, tight running time restrictions, we suspect that there is still some room for improving the approach.

Reproduction of the reported results should be possible by re-running the program. The command line works as follows:

```
npuScheduler.exe A-1.txt A-1_sol_13264537.txt 300 13264537
```

Table 1 Best results and median values over 20 test runs

	Best results		Median			Best results		Median	
Dataset	TPD	TMS	TPD	TMS	Dataset	TPD	TMS	TPD	TMS
`A-1.txt`	1	23	1	23	`B-1.txt`	349	130	357	131
`A-2.txt`	2	41	2	41	`B-2.txt`	481	171	515	178
`A-3.txt`	0	50	0	50	`B-3.txt`	604	214	624	216
`A-4.txt`	65	42	65	42	`B-4.txt`	1283	287	1328	286
`A-5.txt`	153	104	155	106	`B-5.txt`	866	252	909	265
`A-6.txt`	144	94	150	97	`B-6.txt`	1067	246	1098	250
`A-7.txt`	601	206	620	204	`B-7.txt`	827	232	844	241
`A-8.txt`	319	162	340	161	`B-8.txt`	3618	565	3729	574
`A-9.txt`	225	128	240	131	`B-9.txt`	4606	783	4825	806
`A-10.txt`	920	313	955	321	`B-10.txt`	3541	473	3751	474

The program `npuScheduler.exe` is run on instance `A-1.txt` (which has to be stored in the very same directory as the executable). A solution file, `A-1_sol_13264537.txt`, is produced after 300 seconds (in the same directory as the application `npuScheduler.exe`), using a random number seed of `13264537`.

5 Limitations and future research

The proposition of the solution approach assumes that the datasets provided by the organizers of the *MISTA 2013 Challenge* are a representative subset for the problem that should be solved. This implies the following limitations of our concept:

- The generation of feasible mode assignments by means of a randomized algorithm can, as described above, be a problematic issue in case of very limited non-renewable resources. In such situations, infeasible mode selections could be (temporarily) accepted in the search procedure, and evaluated by means of a penalty function [6].
- The implemented neighborhood operators are, with the exception of the SMC, of complexity $\mathcal{O}(n^2)$. In can be expected that larger instances will call for adaptations of the neighborhood search.
  In the *MISTA 2013 Challenge*, the largest test instance was `B-9` with $n = 642$ (incl. dummy-activities). For this particular dataset, no local optimum was found within the given time limit, and thus, no perturbation move was performed here.

Naturally, future research should address the above mentioned limitations.

References

1. Kolisch, R., Drexl, A.: Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions **29**, 987–999 (1997)
2. Kelley, J.E.: The critical-path method: Resources planning and scheduling. In: J.F. Muth, G.L. Thompson (eds.) Industrial Scheduling, pp. 347–365. Prentice Hall, Englewood Cliffs (1963)
3. Demeulemeester, E.L., Herroelen, W.S.: Project Scheduling: A Research Handbook. Kluwer Academic Publishers (2002)
4. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research **130**, 449–467 (2001)
5. Lourenço, H.R., Martin, O., Stützle, T.: Iterated local search. In: F. Glover, G.A. Kochenberger (eds.) Handbook of Metaheuristics, *International Series in Operations Research & Management Science*, vol. 57, chap. 11, pp. 321–353. Kluwer Academic Publishers, Boston, Dordrecht, London (2003)
6. Hartmann, S.: Project scheduling with multiple modes: A genetic algorithm. Annals of Operations Research **102**, 111–135 (2001)

MISTA 2013

A Three-Phases Based Algorithm for the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

The MISTA 2013 Challenge

Federico Alonso-Pecina · Johnatan E. Pecero · David Romero

1 Introduction

In this paper, we address the multi-mode resource-constrained multi-project scheduling problem. We propose a solution based on an hybrid three-phases algorithm: pre-processing, searching and improving, and post-processing. In the first phase, the algorithm computes a feasible initial solution, and performs a naive improvement. In the second phase, we explore the solution space using a Simulated Annealing algorithm. In the third phase, a post-processing step, the algorithm improves the best solution generated in the second phase by means of a Tabu Search procedure.

2 Phase 1. Pre-processing: Getting a feasible solution

In this phase every project $p_i \in P$, where P is a set of n projects, is scheduled respecting time and resource constraints. Each project p_i consists of a set of non-preemptive jobs or activities J_i. For each project p_i, we calculate the set F_i of jobs that are feasible to schedule in the current moment. The first time the only job that we can schedule is the first dummy job J_0, due to precedence constraints. Once the first job is scheduled, we update F_i, and schedule the next feasible job according to a lexicographic order. When we schedule a job, we try to schedule it with a faster mode, always that there is enough resources to programming the unscheduled jobs with at least a mode. In this initialization we assume that all the global resources are available for the project (no other project is scheduled at the same time). After that we get the first feasible solution sol_i of the project, and we calculate a local value TDP_i, which is simply the value of the timing of the last dummy job minus the timing of the first dummy job. Then, we calculate all the jobs that are in the critical path of the project in the solution sol_i, these are the sets of jobs such that exist in the path whose length is equal to TDP_i.

Federico Alonso-Pecina
Universidad Autónoma del Estado de Morelos
E-mail: federicoalonsopecina@hotmail.com

Johnatan E. Pecero
University of Luxembourg
E-mail: johnatan.pecero@uni.lu

David Romero
Universidad Nacional Autónoma de México
E-mail: davidr@matcuer.unam.mx

Later on assume that all the global and local resources are available we try to preassign all the jobs of the critical path in the faster mode, always that exists enough remaining resources to schedule the rest of jobs, at least in a mode. If such preassign is possible, we calculate again a solution of the project p_i, then we obtain a new solution sol_i, we compute the new critical path and try to preassign the new jobs that are part of the critical path. This new preassign is added to the former preassign. We continue this process until it is not possible to preassign jobs and choose the best solution found (the solution with minimal TDP_i) so far by this process. If we are not able to find any solution with this procedure, we assign all the jobs to the slowly modes and obtain a feasible (and with high TDP_i) solution.

After computing a feasible solution to every project p_i, we combine the partial solutions to generate a feasible solution for the whole instance. We start with the first project p_1 and we assign a number to its jobs from 1 to $n_1 - 1$ where n_1 is the number of jobs of the project p_1 and put all the jobs in the solution exactly with its assign. The jobs of the next project p_2 ($j \in p_2$) are numbered from n_1 to $n_1 + n_2$ and set them in the next available timings in such a way that every mode of the job j_1 is respected, with the exception that we need to update the timings. We continue this process until all the projects are gathered in a solution SOL_P.

3 Phase 2. Searching and Improving: Simulated Annealing

Once we obtain a feasible solution we use a Simulated Annealing algorithm to improve the solution. We have used two neighborhoods, the first neighborhood $NEIGH_1$ we simply change the mode m_1 of one random job j_1 to a random mode m_2 *if and only if* the new solution is still feasible. The other neighborhood changes the order that a job is scheduled with all the jobs numbered from 1 to N, where N is the sum of the jobs of all projects. To compute a solution the job number one j_1 is selected and scheduled it as soon as possible, after that we take the next job and we schedule it as earliest as possible, respecting the precedence constraints. If we change the order of the jobs the solution can be different. The second neighborhood $NEIGH_3$ is to change a random job j to another random position *if and only if* the precedence are fulfilled, thus the jobs that are predecessors of j should be with a lower order that j and the successors must have a higher order than j. Before starting the Simulated Annealing process, we perform a run of $10,000$ iterations with random movements of the neighborhood $NEIGH_3$ and we calculate the maximum change in the objective function. The maximum change will set the initial temperature T_0 of the algorithm. The Simulated Annealing is tuned according to the method published in [2]. The maximum length of the Markov chain is set to 7000. In every iteration of the inner cycle, we choose a random neighborhood, after that we choose a random movement and perform it. If the solution is better or equal than the older solution we accept it , otherwise the solution is rejected. When a new solution improves the global solution, the new solution replaces the better global solution. The Simulated Annealing is running since the feasible solution is obtained until reach the final Temperature T_f or reach 20 seconds before the time limit. In this implementation T_f is set to 1.

4 Phase 3. Post-processing: Tabu Search

The best solution computed in the second phase is improved by a Tabu Search algorithm. The Tabu Search algorithm has three neighborhoods $NEIGH_1$, $NEIGH_2$

and $NEIGH_3$. The the algorithm explores the whole neighborhood $NEIGH_1$, if an improvement is made the new solution is accepted. The neighborhood $NEIGH_2$ includes the movements that change the mode m_1 of a job j_1 to another mode m_2 that does not lead to a feasible solution because the movement m_2 lead to a solution that consumes more no renewable resources that the available, but the search tries to reach feasibility searching by another job j_2 to change its mode in such a way that we have enough no renewable resources to perform the mode m_2 of the job j_1. If the solution is improved in the whole neighborhood $NEIGH_1 + NEIGH_2$, then the tabu search seeks in the whole neighborhood $NEIGH_3$. When the global solution is improved the movement is accepted and the movements reach a tabu status for a random number of iterations in the range $[n_i, 2 * n_i]$, where n_i is the number of jobs of the project p_i. If any improvement is made, the tabu search perform the movement that diminishes the less as possible the objective function and does not have the tabu status. The algorithm performs until the time limit is reached or perform on hundred iterations without improvements in the global solution.

Table 1 shows the results for the whole algorithm for 10 runs with the time limit set to 300 seconds. The average value is round to the nearest integer.

INSTANCE	AVERAGE VALUE	BEST VALUE	WORST VALUE
A-1	100, 023	100, 023	100, 023
A-2	300, 042	200, 041	500, 041
A-3	50	50	50
A-4	6, 920, 043	6, 500, 042	8, 300, 044
A-5	17, 690, 108	16, 400, 105	19, 900, 112
A-6	15, 700, 090	14, 100, 092	17, 300, 101
A-7	80, 140, 202	74, 500, 201	84, 900, 203
A-8	45, 520, 164	42, 100, 168	51, 300, 165
A-9	27, 860, 132	25, 100, 127	31, 700, 138
A-10	110, 370, 326	106, 600, 323	119, 700, 328
B-1	40, 310, 133	37, 600, 132	42, 000, 137
B-2	81, 910, 176	79, 000, 168	84, 800, 177
B-3	81, 050, 224	74, 700, 215	88, 300, 229
B-4	178, 830, 291	171, 700, 295	186, 200, 291
B-5	118, 760, 268	111, 900, 268	125, 000, 274
B-6	137, 240, 284	131, 200, 285	144, 200, 274
B-7	101, 440, 235	94, 400, 231	114, 800, 244
B-8	517, 080, 611	503, 500, 620	543, 500, 579
B-9	774, 670, 838	732, 100, 838	815, 100, 846
B-10	477, 510, 503	459, 700, 505	504, 900, 504

Table 1 Value of the solutions of the algorithm for the benchmark.

5 Concluding Remarks

In this work, we addressed the multi-mode resource-constrained multi-project scheduling problem. We proposed an hybrid scheduling algorithm that consists in three phases: pre-processing, searching and improving, and post-processing. First an initial solution is constructed. Then, we refined the initial solution by searching through a neighborhood using a Simulated Annealing search process. Finally, in the post-processing phase, we used a Tabu Search algorithm to improve the best soliton found in the second phase.

References

1. Mista, Mista 2013 challenge: the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem, http://allserv.kahosl.be/mista2013challenge/files/problem-description.pdf, cheking the 28 June, 2013
2. Sanvicente-Sanchez, H. and Frausto, J., Method to Establish the Cooling Scheme in Simulated Annealing Like Algorithms, ICCSA LNCS, Vol. 3095, pages 755-763 (2004)

MISTA 2013

MIP Relaxation and Large Neighborhood Search for a Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

Christian Artigues · Emmanuel Hebrard

1 Introduction

We consider the multi-mode resource constrained multi-project scheduling problem (MMRCMPSP) as proposed by the MISTA 2013 challenge[1]. "Cumulative" resource-constrained project scheduling problems such as the resource-constrained project scheduling problem (RCPSP) with the makespan objective are relatively well-solved by recent methods based on constraint programming (CP) techniques incorporating clause learning [3]. Powerful constraint propagation techniques and tree search-based heuristics are incorporated into commercial CP solvers such as IBM CPOptimizer [1] that obtain competitive results for solving the RCPSP. The additional complexity yielded by mode assignment limited by non-renewable resources make the multi-mode RCPSP a much more challenging problem, although efficient heuristics have been proposed [2]. Furthermore the MISTA 2013 challenge involves a completion time sum criteria! n that raises additional difficulties for CP solvers. In this paper, we precisely consider that the problem hardness comes mainly from the mode assignment and min sum criterion and we propose a large neighborhood search method focusing on these issues, as well as on the presence of local and global resources.

2 The multi-mode resource constrained multi-project scheduling problem

We first briefly present the problem and introduce the notation. A set $\mathcal{P}$ of n projects has to be scheduled on a set of local and global resources. Each project $i \in \mathcal{P}$ has a release date r_i and is made of a set J_i of non preemptive activities. We consider a set of renewable global resources G, and, for each project $i \in \mathcal{P}$, a set of renewable local resources L_i^ρ and a set L_i^ν of local non renewable resources with $L_i = L_i^\rho \cup L_i^\nu$. Each global or local resource has a fixed capacity. c_k denotes the capacity of a global resource $k \in G$ while c_{ik} denotes the capacity of a local resource $k \in L_i$. We denote activity j of project i by A_{ij}. Each activity A_{ij} has a set $M_{ij} = \{1, \ldots, |M_{ij}|\}$ of possible

Christian Artigues,Emmanuel Hebrard
LAAS-CNRS
E-mail: {artigues,hebrard}@laas.fr

[1] http://allserv.kahosl.be/mista2013challenge/

execution modes, such that the selection a mode $m \in M_{ij}$ for activity A_{ij} determine the activity duration and resource requirements. d_{ijm} denote the duration of activity A_{ij} if mode m is selected. r^{ρ}_{ijmk} denote the number of units of a renewable resource $k \in G \cup L^{\rho}_i$ required by activity A_{ij} in mode m with $r_{ijmk} = 0, \forall k \in L^{\rho}_i)$. r^{ν}_{ijmk} denote the number of units of non-renewable resource $k \in L^{\nu}_i \cup G$ required by activity A_{ij} in mode m. For each project i, there is a set E_i of precedence constraints. Let T denote an upper bound of the scheduling horizon. Let s_{ij} denote the start decision variable for activity A_{ij}. Let $m_{ij} \in M_{ij}$ denote the selected mode for activity A_{ij}. For each time $t \in \{0, \ldots, T\}$, let $\mathcal{A}(t) = \{A_{ij} | i \in \mathcal{P}, j \in J_i, s_{ij} \leq t \leq t + d_{ijm_ij} - 1\}$ denote the set of activities in progress at time t. The problem is conceptually formulated as the following lexicographic multi-objective optimization problem where the main objective is the sum of project completion times and the secondary objective is the makespan.

$$MMRCMPSP \qquad Lex(\sum_{i \in \mathcal{P}} c_i, \max_{\in \mathcal{P}} c_i) \tag{1}$$

$$c_i \geq s_{ij} + d_{ijm_{ij}} \qquad \forall i \in \mathcal{P}, \forall j \in J_i \tag{2}$$

$$\sum_{A_{ij} \in \mathcal{A}(t)} r^{\rho}_{ijm_{ij}k} \leq c_k \qquad \forall t \in \{0, \ldots, T\}, \forall k \in G \tag{3}$$

$$\sum_{A_{ij} \in \mathcal{A}(t)} r^{\rho}_{ijm_{ij}k} \leq c_{ik} \qquad \forall t \in \{0, \ldots, T\}, \forall i \in \mathcal{P}, \forall k \in L^{\rho}_i \tag{4}$$

$$\sum_{i \in \mathcal{P}, j \in J_i} r^{\nu}_{ijm_{ij}k} \leq c_{ik} \qquad \forall i \in \mathcal{P}, \forall k \in L^{\nu}_i \tag{5}$$

$$s_{ij} \geq s_{iq} + d_{iqm_{iq}} \qquad \forall i \in \mathcal{P}, \forall (q, j) \in E_i \tag{6}$$

$$s_{ij} \geq 0 \tag{7}$$

The actual objective contains additional constant terms in the sum that have been ignored in this presentation but that we include for presentation of the results.

3 Overview of the method

The method is based on 4 phases. Phase 1 aims at finding a good mode assignment by designing and solving a mixed integer program (MIP) that relaxes the MMRCMPSP. Then the method alternates between 3 phases, each being limited by a maximum number of fails met in the CPOptimizer search tree. Phase 2 fixes the mode assignment $\mathbf{m^*}$ found in phase 1 and builds a CPoptimizer scheduling model by mean of specialized scheduling comments (Interval variables and discrete resources) that is solved during a limited time to obtain a good start schedule $(\mathbf{m^*}, \mathbf{s^*})$. In subsequent phases $(\mathbf{m^*}, \mathbf{s^*})$ is used as a starting point and is updated each time improving solution has been found. In Phase 3, the complete problem is now modeled and solved with CPOptimizer using in addition Alternative interval variables to model mode assignment constraints. Then, Phase 4 consists in a large neighborhood search procedure. At a given iteration, mode assignments are fixed for some activities and released for other activities, based on local and global resource loads considerations as well as on precedence constraint tightness of the current solution. Then the CPOptimizer tree search is launched in a limited time to find an improving solution. The process is iterated until no improvement is reached and the global process restart from phase 2 by fixing the best mode assignment.

We now give two key parts of the above procedure : the relaxed MIP and the large neighborhood search procedure.

4 A relaxed MIP

Non-renewable resource constraints are multi-knapsack constraints that are likely to be well tackled by integer programming solvers. We design a MIP that relaxes the problem by considering only continuous start variables s_{ij} and binary mode assignment variables x_{ijm} where $x_{ijm} = 1$ if mode m is selected for activity A_{ij}. However we completely relax resource constraints and include them any in the objective function. We obtain the following MIP.

$$MIP \qquad \alpha \sum_{i \in \mathcal{P}} \tilde{c}_i + \beta \tilde{C} \tag{8}$$

$$\tilde{C} \geq \sum_{i \in \mathcal{P}, j \in J_i} r^{\rho}_{ijmk} x_{ijm} / c_k \qquad \forall k \in G \tag{9}$$

$$\tilde{c}_i \geq \sum_{i \in \mathcal{P}, j \in J_i} r^{\rho}_{ijmk} x_{ijm} / c_{ik} \qquad \forall i \in \mathcal{P}, \forall k \in L^{\rho}_i \tag{10}$$

$$\tilde{c}_i \geq s_{ij} + p_{ijm} x_{ijm} \qquad \forall i \in \mathcal{P} \tag{11}$$

$$\sum_{i \in \mathcal{P}, j \in J_i} r^{\nu}_{ijmk} x_{ijmk} \leq c_{ik} \qquad \forall i \in \mathcal{P}, \forall k \in L^{\nu}_i \tag{12}$$

$$s_{ij} \geq s_{iq} + d_{iqm_{iq}} \qquad \forall i \in \mathcal{P}, \forall (q,j) \in E_i \tag{13}$$

$$s_{ij} \geq 0 \tag{14}$$

$\tilde{C}$ represents a lower bound on the makespan induced by global constraint load given a mode assignment $\mathbf{m}$ (constraints 9). $\tilde{c}_i$ represent a lower bond on the project completion time which is the max of two components: a lower bound given by the load on the local renewable resources (10) and a lower bound given by the completion time of the project considering only precedence constraints (11,). Non-renewable constraints are not relaxed (12).

Actually $\sum_{i \in \mathcal{P}} \tilde{c}_i$ is the real objective. But C allows to penalize overloading of global resource constraint to compensate the fact that the relaxation of resource constraints does not allow to measure the direct impact on the global resource load on the actual objective.

5 Large neighborhood search

The CPOptimizer model is very efficient for solving the single mode version of the problem. However, adding the choice of mode in the model significantly hinders this efficiency. We therefore opted for a Large Neighborhood Search approach in which we open up a subset of the modes whilst leaving the rest fixed as in the current best solution. We used several strategies to select a pertinent subset of tasks which mode should be opened up. These mode selection phases are alternated with a phase in which the schedule of the fixed-mode RCPSP is optimized.

Project: In the first strategy, we simply leave all mode choices for a single project open. Usually greater returns are expected with projects that come first chronologically, since they condition all subsequent projects. We therefore bias the time spent toward optimizing the modes of early projects rather than late projects.

Adequacy: The second strategy is based on the analysis of the demand profile on each resource. The main idea is that on periods of high usage, "economical" modes should be preferred in order to free up some resources for contending activities, whilst on low usage periods, more greedy modes are made possible.

Given a renewable local or global resource k, we define its *usage* $U_k(t_1, t_2)$ over a time interval $[t_1, t_2]$ as the sum of the total demands over this period. Let $[a, b]$ denote the set of integers greater or equal to a and smaller or equal to b, we have:

$$U_k(t_1, t_2) = \sum_{A_{ij} \ s.t. \ k \in L_i^\rho \cup G} |[t_1, t_2] \cap [s_{ij}, s_{ij} + d_{ijm_{ij}}]| * r_{ijm_{ij}}$$

The usage rate is then defined as the ratio between the usage and the maximum energy that can be supplied by the resource over this period: $U_k(t_1, t_2)/(t_2 - t_1) * c_k$. Moreover, we shall consider the usage rate of activity k during the processing of an activity A_{ij}, denoted $U_{k,ij} = U_k(s_{ij}, s_{ij} + d_{ijm_{ij}})$.

Next we evaluate the resource-efficiency E_{ij} the current mode m_{ij} of an activity A_{ij} by the following ratio:

$$E_{ij} = \frac{max_m(r_{ijm}) - r_{ijm_{ij}}}{max_m(r_{ijm}) - min_m(r_{ijm})}$$

Now, following the initial intuition, the current mode of an activity A_{ij} is seen as "inadequate" if, across resources, the usage ratio of the resource during the execution of A_{ij} is high and A_{ij} is not resource-efficient, or if the usage rate is low and A_{ij} is resource-efficient. In order to rank activities we score them by the average absolute difference between usage rate and the energy "inefficiency" (i.e., $1 - E_{ij}$). that is, the score of an activity A_{is} is:

$$S(A_{ij}) = \frac{\sum_{k \in L_i^\rho \cup G} |U_{k,ij} + E_{ij} - 1|}{|L_i^\rho \cup G|}$$

Period: Activities that overlap in time directly interact with each other. We can therefore bias the choice of tasks which mode we will open up, by their position in time. We arbitrarily divide the scheduling horizon into $n/2$ periods of equal length. Then for each period, we select up to n activities with highest inadequacy score $S(A_{ij})$. As in the first strategy we bias the search effort toward the first period.

Decay: To diversify the set of activities on which we search for the best mode allocation, we add a geometrically decaying term to the score $S(A_{ij})$. This term was arbitrarily set to the value 1. Given an activity which mode has been opened up, its weight is decreased by 10% whenever a improving solution that does not assign this task to a different mode is found. Moreover, when no improving solution is found, all tasks which mode was opened see their weights decreased twice by 10%.

6 Experimental Results

We ran our approach on a machine running Linux on a 3.4GHz with 4GB of ram with a time cutoff of 5 minutes. We report the objective value found within these 5 minutes for each instance of set A and B in Table 1.

Table 1 Results on set A and B

Instance	Objective
A-1.txt	100023
A-2.txt	200041
A-3.txt	50
A-4.txt	6500042
A-5.txt	16300106
A-6.txt	16300087
A-7.txt	64300194
A-8.txt	30500146
A-9.txt	23400116
A-10.txt	108700304
B-1.txt	43600129
B-2.txt	52600156
B-3.txt	66300207
B-4.txt	141600281
B-5.txt	103600249
B-6.txt	109700217
B-7.txt	90700230
B-8.txt	360800524
B-9.txt	552400747
B-10.txt	405300429

7 Conclusion

As part of the "Mista 2013 Challenge", we introduced a MIP and CP based approach for solving the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem. This approach relies on a standard CP model for the single mode problem, and on a MIP relaxation in order to find a good initial mode allocation. Moreover, we use Large Neighborhood Search alternating between mode selection and fixed-mode schedule optimization.

References

1. IBM ILOG CPLEX Optimization Studio 12.5, reference manual
2. V. Van Peteghem, M. Vanhoucke. Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. Journal of Heuristics 17:705–728, 2011.
3. A. Schutt, T. Feydy, PJ Stuckey, MG Wallace. Explaining the cumulative propagator. Constraints16(3):250–282, 2011

MISTA 2013

Automatic Design of a Hybrid Iterated Local Search for the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

Manuel López-Ibáñez · Franco Mascia · Marie-Eléonore Marmion · Thomas Stützle

1 Introduction

This paper details our submission to the MISTA 2013 challenge, which deals with the multi-mode resource-constrained multi-project scheduling problem (MRCMPSP). Kolisch and Hartmann [4] recommend metaheuristics as the best-performing methods when tackling the resource-constrained project scheduling problem with a single project and without multiple-modes. Most of these metaheuristics share many similar components, such as neighborhoods used in local search, schedule generation procedures, etc. Given the number of possible combinations of algorithmic components and the various ways in which simple metaheuristics can be combined into a single algorithm, finding the right combination for the MRCMPSP would be, in principle, an arduous task of experimentation by trial-and-error.

Instead, we used a recent automatic method [10], which combines (*i*) a description (given as a grammar) of the space of potentially valid algorithms for a problem and (*ii*) a method for searching the best algorithm by instantiating algorithms from this grammar. The approach shares some similarities with genetic programming (GP) [11] and grammatical evolution (GE) [3], yet there are crucial differences. First, GP/GE typically attempt to generate programs from very basic components, whereas our approach relies on humans to provide problem-specific components for the particular problem. Second, GP/GE use a tree-based and a codon-based representation, respectively, to instantiate programs from the grammar. Instead, we use a parametric representation, that is, the grammar description is transformed into a number of categorical and numerical parameters, and some of them might be only enabled depending on other parameters (conditional parameters). This transformation requires to specify the maximum number of times that each rule in the grammar can be applied (similar restrictions exist with other representations, given the existence of recursive derivation rules). Finally, GP/GE use evolutionary algorithms to search for the best instantiation of the grammar. Our approach, by contrast, relies on irace [6], an automatic configuration tool typically used for offline parameter tuning. The characteristics of irace makes it

Manuel López-Ibáñez, Franco Mascia, Marie-Eléonore Marmion, and Thomas Stützle
IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium
E-mail: {mmarmion;fmascia;manuel.lopez-ibanez;stuetzle}@ulb.ac.be

ideal to handle complex parameter spaces, with categorical, numerical and conditional parameters. Moreover, irace is designed for handling heterogeneous problem instances.

2 Automatic Bottom-up Design of Hybrid ILS Algorithms

The grammar (in Backus-Naur Form) used in this work is divided into a problem-independent part and a problem-specific part. Following previous work [9], the problem-independent grammar (Fig. 1) generates hybrid iterated local search (ILS) algorithms. The main algorithm starts with a problem-specific initialization followed by an ILS. Each ILS is described by four components, a perturbation, a subsidiary local search (LS), an acceptance criterion and a time ratio for the subsidiary LS. The subsidiary LS may be either an ILS, a simulated annealing (SA) or an iterative descent LS using a problem-specific neighborhood. The iterative descent may be either a best-improvement one (*BestImpr*), a first-improvement (*FirstImpr*) one, or a first-improvement descent that continues the search at the same position in the neighborhood after moving to an improving neighbor (*FirstImprCont*). The acceptance criterion determines the solution that replaces the current solution in the ILS. All acceptance criteria accept a better solution but they may also accept a solution with equal or worse objective value in some circumstances. The ones used here are well-known acceptance criteria, their precise definition can be found in our previous work [9]. Some of them have extra parameters. For example, *Cooling* accepts solutions according to the Metropolis criterion, that is, by biasing a probabilistic decision according to a temperature parameter that is adjusted according to a cooling schedule. Finally, each call to the subsidiary LS is given a time limit $t_{\text{subls}} = t_{\text{ratio}} \cdot t_{\text{ls}}$, where t_{ls} is the time limit of the caller and t_{ratio} is a parameter. Each LS is also limited by the maximum time assigned to the whole algorithm.

In order to generate an algorithm for the MRCMPSP from this grammar, we need to implement the required problem-specific components. In our implementation, the representation of schedules is: (i) a standard permutation of the activities, where each activity always appears later than any of its predecessors and earlier than any of its successors, and (ii) a vector of modes, where each entry is the execution mode assigned to each activity. We also apply a well-known pre-processing step in order to reduce the search space [12]. Other MRCMPSP-specific components of our algorithm are configurable, and, thus, they are shown in the grammar of Fig. 2.

Evaluating a solution requires applying a schedule generation (SG) procedure. Unless specified otherwise in the grammar, we use the forward serial SG for a precedence feasible activity list [2]. This procedure is faster than the classical serial SG but produces semi-active schedules instead of active ones [13]. In some specific cases, the SG used is configurable `<ps:sg>`: *ForwardSerial* applies the forward serial SG, *Forward-BacwardsSerial* applies the same procedure forward and backwards and keeps the best solution found, and *multi-mode backward-forward* (MM-BF) generates a schedule using the forward serial SG and then applies MM-BF [8] until no improvement is found.

Initialization constructs one or more solutions heuristically and keeps the best solution constructed. Each solution is constructed by a parallel SG (c.f. Brooks's parallel method [1]) using a heuristic for selecting activities and another for selecting modes. The constructed solutions are re-scheduled using one of the three SGs in the hope that they can be further improved.

The heuristics for selecting activities include the well-known RANDOM-ACTIVITY, the static late finish time (MIN-LFT), static feasible slack (MIN-FSLK), dynamic feasible

```
 <algorithm> ::= <ps:initialization> <ils>
       <ils> ::= ILS(<ls>, <accept>, <ps:perturb>, <time_ratio>)
        <ls> ::= <ils> | <sa> | <descent>(<ps:neighborhood>)
   <descent> ::= BestImpr | FirstImpr | FirstImprCont
       <sa>  ::= SA(<ps:perturb>, <cooling_schedule>)

    <accept> ::= Always | Improves | ImprovesEqual | Prob(<prob>) | ProbRandom
               | Threshold(<threshold>) | Metropolis(<cooling_sched>)
      <prob> ::= [0, 1]
 <threshold> ::= [0, 1]
<cooling_sched> ::= Cooling(<init_temp>, <final_temp>, <decr_temp_ratio>, <span>)
     <init_temp> ::= {1, 2,..., 10000}
    <final_temp> ::= {1, 2,..., 100}
<decr_temp_ratio> ::= [0, 1]
          <span> ::= {1, 2,..., 10000}
    <time_ratio> ::= {0.1, 0.2, ..., 1}
```

Fig. 1 Problem-independent grammar for generating hybrid ILS algorithms.

```
            <ps:sg> ::= ForwardSerial | ForwardBackwardSerial | multimodeFBSerial
<ps:initialization> ::= SGforInit(<ps:sg>) <ps:initsols>
      <ps:initsols> ::= <ps:init_heur>
                      | <ps:init_heur> <ps:initsols>
     <ps:init_heur> ::= init(<ps:activity_heu>, <ps:mode_heu>)
  <ps:activity_heu> ::= RANDOM_ACTIVITY | MIN_LFT | MIN_FSLK | TWK_LST | TWK_EST
                      | TWK_GDR | MAXTWK_EST | MAXTWK_LST | SASP
      <ps:mode_heu> ::= RANDOM_MODE | MIN_MNR | MIN_DUR | BEST_MODE
  <ps:neighborhood> ::= ModeNeighborhood | ActivityNeighborhood
       <ps:perturb> ::= PerturbFeasibleMode(<ps:perturb_pct>)
                      | PerturbActivity(<ps:perturb_pct>)
                      | PerturbReleaseManyModes(<ps:perturb_pct>)
                      | PerturbHeuRandMode(<ps:activity_heu>, <ps:sg>)
                      | PerturbHeuRandActivity(<ps:mode_heu>, <ps:sg>)
   <ps:perturb_pct> ::= {0, 1, ..., 100}
```

Fig. 2 Problem-specific part of the grammar.

slack (MIN-DFSLK), and shortest activity from shortest project (SASP) [5]. In addition, MAXTWK-EST, MAXTWK-LST, TWK-LST, TWK-EST and TWK-GDR all use the concept of total work content (TWK) [5]. Both MAXTWK-EST and MAXTWK-LST are applied to all schedulable activities at once and they differ on how they break ties; TWK-LST, TWK-EST, and TWK-GDR first select a project according to TWK, and then select an activity according to a secondary rule [7]. For computing these heuristic values in the multi-mode case, we consider all feasible modes. In addition, we save the mode used by the activity heuristic so it can be selected later (by BEST-MODE).

The heuristics for selecting modes are random (RANDOM-MODE), minimum normalized resources (MIN-MNR) [8], a dynamic version of MIN-MNR that takes into account the resources available in the current solution being constructed (MIN-DMNR), minimum duration (MIN-DUR, also called shortest feasible mode, which selects the feasible mode with the shortest duration), and BEST-MODE, which is a heuristic that either selects whatever mode was chosen by the activity heuristic, if any, otherwise, it selects the shortest feasible mode (MIN-DUR).

We consider two neighborhoods in our iterative improvement algorithms. The mode neighborhood simply replaces the mode of each activity with a different mode that respects the non-renewable constraints, considering the resources released by the discarded mode and required by the other activities [2]. The shift (insertion) neighborhood of activities removes one activity from the activity list and reinserts it in a position between its latest predecessor and its earliest successor [2].

```
SGforInit(MM-BF)
Init(MIN-FSLK, MIN-DUR)
Init(TWK-EST, MIN-DMNR)
// Additional heuristics manually added (Fig. 4)
ILS(FirstImprCont(ModeNeighborhood),
    ImprovingEqual, perturbActivity(0%),
    t_subls = 1)
```

Fig. 3 Algorithm instantiated from the grammar and submitted to the 2013 MISTA Challenge.

```
Init(RANDOM-ACTIVITY, MIN-MNR)
Init(MIN-LFT, MIN-MNR)
Init(TWK-EST, RANDOM-MODE)
Init(TWK-EST, BEST-MODE)
Init(TWK-LST, MIN-DMNR)
Init(TWK-LST, BEST-MODE)
```

Fig. 4 Additional initialization heuristics manually added to the algorithm.

We implemented four perturbation methods: *PerturbFeasibleMode* uses the mode neighborhood to randomly change the mode of a number of activities, *PerturbActivity* performs a shift move on a number of random activities, *perturbReleaseManyModes* releases the non-renewable resources assigned to a number of random activities all at once, then tries to assign new random modes to those activities without violating the non-renewable resource constraints. These methods only modify a percentage of the activities. By contrast, *PerturbHeuRandMode* and *PerturbHeuRandActivity* generate a completely new solution by using the initialization method discussed above with either the RANDOM-MODE or the RANDOM-ACTIVITY heuristic, respectively. These two latter perturbations are similar to a restart mechanism.

3 Our algorithm for the MISTA 2013 Challenge

In order to find the best instantiation of the grammar described above, we run irace using as training instances the two benchmark sets provided by the MISTA 2013 Challenge, and with a maximum budget of 50 000 runs (each run evaluates an instantiation of the grammar on a single instance for 300 seconds). During this training phase, each instantiation of the grammar is a sequential algorithm. The best instantiation found by irace in shown in Fig. 3.

Although we set a limit of two derivations per rule, which would have allowed for two levels of ILS, the algorithm generated by irace is simpler than what we expected (Fig. 3). It is an ILS that uses a *FirstImprCont* descent applied to the mode neighborhood as the subsidiary local search. This descent does not have a time limit: it continues until no improvement is found (or the overall time limit is reached). If the solution generated by the descent is better or equal than the initial one, it replaces the initial one as the current solution. The perturbation is then applied to the current solution, and it modifies one random activity (0%, but the minimum is one activity) by reinserting it in a different position in the schedule. Our intuition is that for very large instances, the best solution found is continually being improved, and that additional levels in the ILS do not speed up this convergence.

In our experiments, we noticed that generating more than two initial solutions, using different heuristic rules, leads to better results and requires little extra computation time. Hence, we manually added six additional initializations (Fig. 4). Thus, the final algorithm generates eight initial solutions and passes the best one to the ILS procedure.

Finally, we embed this ILS algorithm within a multi-threaded solver, which runs the algorithm four times in parallel with different random seeds and returns the best solution found. We compile this algorithm with GCC 4.4.6 with optimization level -O3 and run it 15 times (with different random seeds) on each problem instance provided by challenge organizers. Each run is stopped after 300 wall-clock seconds. Experiments

Table 1 Mean and standard deviation of the total objective cost obtained by 15 independent runs (with different random seed) of our proposed algorithm on each problem instance provided by the challenge organizers.

Instance	# Projects	# Activities†	Mean	Sd
A-1	2	20	100023.0	0.0
A-2	2	40	200041.0	0.0
A-3	2	60	6716.7	25820.9
A-4	5	50	7733384.6	308609.9
A-5	5	100	19326779.4	471271.0
A-6	5	150	18906768.8	517503.1
A-7	10	100	65733541.5	952942.3
A-8	10	200	41460167.7	1202855.6
A-9	10	300	34213480.4	720980.5
A-10	10	300	121873675.7	2599030.5
B-1	10	100	47513470.0	993455.0
B-2	10	200	62533505.2	1478739.0
B-3	10	300	77833562.7	2689041.3
B-4	15	150	171953646.1	2329583.1
B-5	15	300	125606943.5	2904055.1
B-6	15	450	147626938.3	3769328.2
B-7	20	200	117420271.2	1729658.8
B-8	20	400	430687283.1	9350930.5
B-9	20	600	641294225.6	6006603.6
B-10	20	420	456520505.2	6424312.1

† not including dummy activities.

were performed on a AMD Opteron 6272 (16 cores with 2.1GHz, 16MB L2/L3 cache) with 64GB RAM running under Cluster Rocks Linux version 6/CentOS 6.3, 64bits. Each run only uses 4 cores at most. Table 1 reports the mean and standard deviation of the total objective cost, computed as $\text{TPD} \cdot 10^5 + \text{TMS}$, over the 15 independent runs performed on each instance.

4 Conclusions

In this extended abstract, we only report the results obtained on the instances provided by the challenge organizers, but it is our intention to carry out a complete experimental analysis of the proposed algorithm and compare it with other proposals.

The automatic design method helped us to develop an effective algorithm in a very short time with no previous experience in project scheduling. Moreover, if new algorithmic components that are expected to improve the results become available, re-generating a new hybrid ILS algorithm only requires computing effort. The same method could be used to design other hybrid metaheuristics for the MRCMPSP.

Acknowledgements The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 246016. Marie-Éléonore Marmion acknowledges support from the ERCIM "Alain Bensoussan"

Fellowship Programme. This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium and the FRFC project *Méthodes de recherche hybrides pour la résolution de problèmes complexes*. This research has also received funding from the COMEX project within the Inter-university Attraction Poles Programme of the Belgian Science Policy Office. Manuel López-Ibáñez, Franco Mascia and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are postdoctoral researchers and a research associate, respectively.

References

1. Bedworth, D.D., Bailey, J.E.: Integrated Production Control Systems: Management, Analysis, Design, vol. 2. John Wiley & Sons, New York, NY (1982)
2. Bouleimen, K., Lecocq, H.: A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research **149**(2), 268–281 (2003)
3. Burke, E.K., Hyde, M.R., Kendall, G.: Grammatical evolution of local search heuristics. IEEE Trans. Evol. Comput. **16**(7), 406–417 (2012)
4. Kolisch, R., Hartmann, S.: Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research **174**(1), 23–37 (2006)
5. Kurtulus, I., Davis, E.W.: Multi-project scheduling: Categorization of heuristic rules performance. Management Science **28**(2), 161–172 (1982)
6. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
7. Lova, A., Tormos, P.: Analysis of scheduling schemes and heuristic rules performance in resource-constrained multiproject scheduling. Annals of Operations Research **102**(1-4), 263–286 (2001)
8. Lova, A., Tormos, P., Cervantes, M., Barber, F.: An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. International Journal of Production Economics **117**(2), 302–316 (2009)
9. Marmion, M.E., Mascia, F., López-Ibáñez, M., Stützle, T.: Automatic design of hybrid stochastic local search algorithms. In: Blesa, M.J., et al. (eds.) Hybrid Metaheuristics, *LNCS*, vol. 7919, pp. 144–158. Springer, Heidelberg (2013)
10. Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T.: From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness. In: Learning and Intelligent Optimization, LION 7, *LNCS*, vol. to appear. Springer, Heidelberg, (2013)
11. Mckay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based genetic programming: A survey. Genetic Programming and Evolvable Machines **11**(3-4), 365–396 (2010)
12. Sprecher, A., Hartmann, S., Drexl, A.: An exact algorithm for project scheduling with multiple modes. OR Spektrum **19**(3), 195–203 (1997)
13. Sprecher, A., Kolisch, R., Drexl, A.: Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. European Journal of Operational Research **80**(1), 94–102 (1995)

MISTA 2013

A simple stochastic local search for multi-mode resource-constrained multi-project scheduling

Leonardo M. Borba · Alexander J. Benavides · Tadeu Zubaran · Germano M. Carniel · Marcus Ritt

Abstract In the multi-mode resource-constrained multi-project scheduling (MMRCMPSP) problem we have to find execution modes and starting times for the jobs of multiple projects, such that the total project delay is minimized. The delay of a project is the difference between its makespan and a lower bound on its duration. A secondary objective is to minimize the total makespan of all projects. The jobs of a project compete for renewable and non-renewable local resources, and for global renewable resources shared among the projects. The duration and resource requirements of a job depend on its execution mode.

We present a simple stochastic local search method for this problem. It maintains a feasible permutation and mode selection of the jobs, and improves it by swapping the order of two jobs, or changing the mode of a job. We propose several techniques to reduce size of the analyzed neighborhoods. The method has ranked fourth during the qualification phase of the MISTA challenge, and was qualified for the second phase. We present computational experiments on the twenty instances of the challenge and compare them to the results of the qualification phase.

1 Introduction

In the MMRCMPSP (multi-mode resource-constrained multi-project scheduling problem) we have to find a common schedule for multiple projects. Each project consists of several jobs which have to be scheduled in a given partial order after the projects' release time. Each job can choose among multiple execution modes of different processing time and resource requirements. Resources may be local to a project, or global, i.e. shared among the projects. Local resources may be renewable, with a fixed capacity per time, or non-renewable, with a given total capacity. Global resources are always renewable.

Formally, let $\mathcal{P} = [n]$ be the set projects, $J_i = [m_i]$ the set of jobs, and r_i the release time of project $i \in \mathcal{P}$. The jobs in J_i have to satisfy a partial order $\preceq$ on J_i. Each project $i \in \mathcal{P}$ has ρ_i local renewable resources of capacity c_{ij}, $j \in [\rho_i]$ and $\overline{\rho}_i$ local non-renewable resources of capacity $\overline{c}_{ij}$, $j \in [\overline{\rho}_i]$. Further, there are ρ global renewable resources of capacity c_j, $j \in [\rho]$. A job $j \in J_i$ has μ_{ij} execution modes. In execution mode $m \in [\mu_{ij}]$ it has processing time

Leonardo M. Borba · Alexander J. Benavides · Marcus Ritt
Universidade Federal do Rio Grande do Sul, Instituto de Informática
E-mail: {leonardo.borba,ajbenavides,tkzubaran,gmcarniel,marcus.ritt}@inf.ufrgs.br

p_{ijm}, local renewable resource requirements r_{ijmk}, $k \in [\rho_i]$, local non-renewable resource requirements $\overline{r}_{ijmk}$, $k \in [\overline{\rho}_i]$, and global renewable resource requirements r^g_{ijmk}, $k \in [\rho]$.

A solution of the MMRCMPSP is a selection of starting times s_{ij} and modes $m_{ij} \in [\mu_{ij}]$, $i \in \mathcal{P}$, $j \in J_i$. Let $f_{ij} = s_{ij} + p_{ijm_{ij}}$ be the finishing time of job $j \in J_i$ in project $i \in \mathcal{P}$, and

$$R_{irt} = \sum_{\substack{j \in J_i \\ t \in [s_{ij}, f_{ij}]}} r_{ijm_{ij}r}, \qquad \forall i \in \mathcal{P}, r \in [\rho_i], t \in [0, T],$$

$$R^g_{rt} = \sum_{i \in \mathcal{P}} \sum_{\substack{j \in J_i \\ t \in [s_{ij}, f_{ij}]}} r^g_{ijm_{ij}r}, \qquad \forall r \in [\rho], t \in [0, T]$$

be the local renewable resource of resource r in project i at time t, and the global renewable requirements of resource r at time t, respectively. Let further

$$\overline{R}_{ir} = \sum_{\substack{j \in J_i \\ t \in [s_{ij}, f_{ij}]}} \overline{r}_{ijm_{ij}r}, \qquad \forall i \in \mathcal{P}, r \in [\overline{\rho}_i]$$

be the local non-renewable requirement of resource r in project i. Then, a valid solution satisfies the precedence constraints

$$f_{ij} \leq s_{ij'}, \qquad \forall i \in \mathcal{P}, j, j' \in J_i, j \preceq j', \tag{1}$$

the release time constraint

$$r_i \leq s_{ij}, \qquad \forall i \in \mathcal{P}, j \in J_i, \tag{2}$$

and the resource constraints

$$R_{irt}, \leq c_{ir}, \qquad \forall i \in \mathcal{P}, r \in [\rho_i], t \in [0, T], \tag{3}$$

$$\overline{R}_{ir}, \leq \overline{c}_{ir}, \qquad \forall i \in \mathcal{P}, r \in [\rho_i], \tag{4}$$

$$R^g_{rt}, \leq c^g_r, \qquad \forall r \in [\rho], t \in [0, T], \tag{5}$$

for some upper bound on the total time horizon of any valid schedule T (e.g. $T = \max_{i \in \mathcal{P}} r_i + \sum_{i \in \mathcal{P}} \sum_{j \in J_i} \max_{m \in [\mu_{ij}]} p_{ijm}$).

The objective of the MMRCMPSP is to minimize the total project delay $TPD = \sum_{i \in \mathcal{P}} PD_i$. The delay PD_i of project i is the difference between its makespan $MS_i = f_i - r_i$, where $f_i = \max_{j \in J_i} f_{ij}$ and its critical path duration $CP_i = \max_{j \in J_i} e_{ij}$ for earliest finishing times

$$e_{ij} = \min_{m \in [\mu_{ij}]} p_{ijm} + \max_{j' \in J_j | j' \preceq j} e_{ij'}$$

(here we take the maximum of the empty set to be 0). If there are several solutions of the same total project delay, the total makespan $TMS = \max_{i \in \mathcal{P}} f_i - \min_{i \in \mathcal{P}} r_i$ serves as a tie breaker.

2 A stochastic local search for the MMRCMPSP

We propose a simple stochastic local search for the MMRCMPSP. It pre-processes the problem, creates an initial solution by solving each project individually and merging these solutions into a single solution for the whole problem, applies a local search and two variants of an (iterated) stochastic local search in parallel to improve the solution. In the following, these steps are described in more detail.

2.1 Pre-processing

In a pre-processing step dominated and unsatisfiable modes are removed. A mode is *dominated* if there is another mode whose execution time is not longer and whose resource requirements are not larger. A mode is *unsatisfiable* if some renewable resource requirement exceeds the capacity. For the instance set of the challenge the pre-processing has a limited effect: there are no dominated modes, and only four instances, namely A4, B1, B4, and B7, have between one to nine unsatisfiable modes.

2.2 Representation and evaluation

A solution to the MMRCMPSP is represented by a permutation of the jobs and a mode selection for each job. From this, a complete, feasible solution is constructed by processing each job in the order of the permutation, starting it at the earliest time after the termination of all its predecessors which is able to satisfy its resource requirements during its execution. For the construction we use a data structure that stores only time points at which the capacity of some renewable resource changes, and maintains them ordered by time. The construction needs time $O(m^2k)$ and space $O(mk)$ for a project with m jobs and k renewable resources.

2.3 Initial solution

We guarantee during the search that every solution representation leads to a valid solution of the problem. This is achieved by permitting only permutations that satisfy the precedence constraints (1) and mode selections that satisfy the non-renewable resource constraints (4).

An initial feasible solution satisfying constraints (1) and (4) is obtained as follows. The permutation is computed by a topological sorting, which chooses in every step a random minimal element of the remaining (not yet scheduled) jobs. Of all mode selections which satisfy (4) we choose one of shortest total processing time. This mode selection is found by dynamic programming: For a project with m jobs and $\overline{\rho}$ non-renewable resources, let $S(i,c)$ be the smallest total processing time of jobs $i, i+1, \ldots, m$ given non-renewable resources $c \in \mathbb{Z}^{\rho}$. Let μ_i be the number of modes of job i, p_{ik} its processing time in mode k and $r_{ik} \in \mathbb{Z}^{\rho}$ its non-renewable resource requirements in mode k. The shortest total processing times satisfy the recurrence

$$S(i,c) = \begin{cases} 0 & \text{if } i > m \\ \infty & \text{if } r_{ik} \not\leq c \text{ for all } k \in [\mu_i] \\ p_{ik} + S(i+1, c-r_i) & \text{otherwise, where } k = \operatorname{argmin}_{k' \in [\mu_i] \mid r_{ik} \leq c} p_{ik'} \end{cases},$$

and the smallest total processing time is $S(1,c)$ for capacities c. The recurrence can be computed by dynamic programming in time $O(m\rho \max_{i\in[m]} \mu_i \prod_{i\in[\rho]} c_i)$ and space $O(m \prod_{i\in[\rho]} c_i)$.

2.4 Local search

Our local search uses two types of moves: a swap move, that interchanges two jobs of the current permutation, and a mode move, that changes the mode of some job. To maintain the feasibility of the solution, as described above, only moves that satisfy the precedence

Table 1 Parameter settings used in the computational experiments.

Sample size s in sample stochastic local search	$\min\{\max\{25, m^2/10\}, 250\}$
Maximum rank difference δ for swap moves	$m/10$
Number of trials T until convergence in stochastic local search	$m^2/2$
Number of trials T_0 that accept only improving moves	$T/100$

constraints (1) and non-renewable resource constraints (4), are permitted. For the current solution we maintain the minimal and maximal permissible rank of each job in the permutation, as well as the total consumption of non-renewable resources. This allows to check the validity of a swap move in time $O(1)$ and that of a mode move in time $O(\overline{p})$. To limit the size of the neighborhood, swap moves are only considered between jobs whose rank in the current permutation differs by at most δ.

We use three local search algorithms in our approach. The basic first improvement local search visits systematically all swap and mode moves, and repeatedly accepts the first improving move, or stops if no such move exists. The stochastic local search repeatedly chooses a random move, and checks if it is acceptable. It stops after at most T unsuccessful trials. A move is accepted if it improves the current solution during the first $T_0 \leq T$ trials, or if it does not worsen the solution in the remaining trials. Finally, the sample stochastic local search chooses a sample of s of random moves without replacement, and processes them in order of non-decreasing change in objective function value. The acceptance and stopping criteria are as in the stochastic local search. The sample is maintained during several iterations, until all moves have been consumed, and then recreated with moves from the current solution.

The complete search proceeds in three steps. In the first step each project of the initial solution is optimized individually by applying the sample stochastic local search. Next, the first improvement local search is applied. Finally, the search continues with one of the two stochastic local searches. If one these terminates before the time limit, a random permutation satisfying (1) is generated, and the search continues from this solution, maintaining the current mode selection. The final version uses multiple threads, where half of the threads executes to stochastic local search in the third step, and the other half the sample stochastic local search.

3 Computational results

We implemented our heuristic in C++ and evaluated it on a PC with an Intel Core i7 930 processor running at 2.8 GHz and with 12 GB of main memory. We used the two sets of 10 instances (A and B) provided during the challenge. More information on the instances is available in Kinable, Smet, Vancroonenburg, Berghe, Verstichel, and Wauters [1]. After some preliminary experimentation, we chose the parameters shown in Table 1.

Table 2 presents the results of twelve replications with different random seeds on a single processor. For both variants of the stochastic local search we present the mean objective function value, its standard deviation, the best value found in all 12 runs, and the number of repetitions of the local search. Best values are highlighted in bold. We can see that the two variants obtain similar results. Both are reasonably robust, having small standard deviations. Both find the best known values from the qualification phase only for the smallest three instances, and have relative deviations from about 7% to 40% on the remaining instances. However, these values improved over the qualification phase, and would rank the method

Table 2 Results for a single-processor execution: mean, standard deviation and best value over 12 replications, and the number of repetitions.

Inst.		Stochastic local search				Sample stochastic local search			
		Avg	±sd	Best	Rep.	Avg	±sd	Best	Rep.
A	1	1.0	0.0	1.0	36506.2	1.0	0.0	1.0	36269.2
A	2	2.0	0.0	2.0	3840.2	2.0	0.0	2.0	3813.7
A	3	0.0	0.0	0.0	1413.0	0.0	0.0	0.0	1414.5
A	4	69.7	2.5	66.0	902.9	**69.4**	2.2	67.0	891.0
A	5	196.4	7.3	188.0	67.6	**195.8**	7.6	185.0	65.6
A	6	158.8	9.2	147.0	16.1	**157.5**	3.9	151.0	16.8
A	7	789.6	19.1	764.0	45.0	**783.1**	13.4	749.0	44.4
A	8	422.1	16.3	389.0	4.8	**410.2**	16.3	387.0	5.1
A	9	**271.2**	19.9	241.0	1.6	306.3	41.7	261.0	2.5
A	10	**1049.8**	41.3	994.0	1.9	1062.9	59.5	996.0	2.0
B	1	**390.5**	4.1	381.0	36.0	390.6	3.4	384.0	35.9
B	2	813.1	18.3	775.0	4.8	**811.0**	14.5	794.0	4.6
B	3	739.3	23.8	696.0	1.9	**754.3**	27.5	706.0	2.1
B	4	**1859.0**	44.6	1787.0	13.4	1867.4	32.7	1813.0	13.5
B	5	**1193.9**	65.0	1103.0	1.3	1326.8	61.0	1205.0	2.0
B	6	**1368.1**	48.8	1295.0	1.0	1409.0	35.3	1339.0	2.0
B	7	1093.8	76.7	944.0	7.5	**1092.5**	48.1	1023.0	8.1
B	8	**4897.8**	105.0	4761.0	1.0	5003.4	89.9	4862.0	2.0
B	9	8563.3	195.3	8301.0	1.0	**8562.8**	173.9	8342.0	2.0
B	10	4757.4	79.7	4646.0	1.0	**4714.8**	101.3	4536.0	2.0

fourth or better, showing a great variance among the competitors. We can also observe that for the larger instances, both stochastic local searches make very few repetitions, and in some cases do not even converge during the time limit of 300s.

4 Conclusions

We have proposed a heuristic for the MMRCMPSP based on simple stochastic local searches, combined with some techniques to reduce the effort of the neighborhood search. The method turned out to be competitive in the qualification, and has been improved further. We believe the short time limit of 300s favours simple methods, and our approach may be competitive in the final round. Its main limitation is the slow convergence, which could be improved by more directed methods of overcoming (non-strict) local minima.

References

[1] Joris Kinable, Pieter Smet, Wim Vancroonenburg, Greet Vanden Berghe, Jannes Verstichel, and Tony Wauters. *MISTA 2013 Challenge; the multi-mode resource-constrained multi-project scheduling problem*. 2013.

MISTA 2013

Evolutionary algorithm and post-processing to minimize the total delay in multi-project scheduling

Hermann Bouly · Duc-Cuong Dang · Aziz Moukrim · Huang Xu

Introduction

This abstract presents our approach to solve the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem (MRCMPSP) in which the main objective is to minimize the Total Project Delay (TPD), in the context of the MISTA 2013 Challenge. The described algorithm is the outcome of our experiment-driven development over the problem instances provided along with the challenge. A short description of the scheduling problem is given in the first section, then our model and the Evolutionary Algorithm (EA) are described. Next, the local search (LS) to improve the solution of the EA is presented. Finally, our experimental results are discussed in the last section.

1 Challenges of MRCMPSP

An exhaustive description of the problem is available at [5]. MRCMPSP is a generalization of the well-known Resource Constrained Project Scheduling Problem (RCPSP) (see Chapter 1 of [1]). Finding the optimal starting date for each job of RCPSP is generally NP-hard [3]. In MRCMPSP, many projects are involved and each one could be an independent instance of RCPSP with a specific release date. The execution of those projects requires renewable and non-renewable resources. The resources are limited and some renewable ones are shared between projects. In addition, each job of MRCMPSP can have several modes of execution. The need to make choice between

Hermann Bouly
E-mail: hermann.bouly@gmail.com

Duc-Cuong Dang
School of Computer Science, University of Nottingham, UK
E-mail: duc-cuong.dang@nottingham.ac.uk

Aziz Moukrim
Heudiasyc, UMR 7253 CNRS, Université de Technologie de Compiègne, France
E-mail: aziz.moukrim@hds.utc.fr

Huang Xu
E-mail: benjxu@gmail.com

those modes, meaning different durations and resource demands for each job, significantly increases the size of the search space. The available modes of a job are generally non-dominated, e.g. modes demanding a lot of resources tend to have fast execution. Finally, two objective criteria are considered in hierarchical order: (i) the Total Project Delay (TPD) and (ii) the Total Makespan (TMS). Those two objectives are unrelated and one can be contrary to the other.

2 Evolutionary algorithm

In our approach, a solution is indirectly represented by a *sequence* of triplets $\langle p, j, m \rangle$, referred as *mode keys*, so that each job j of a project p appears exactly once and m is the choosen mode. We adapted the event-based model and the Serial Scheduling Generation Scheme (SGS) (see Chapter 6 of [1]) for building schedules of MRCMPSP from these sequences. The adaption for MRCMPSP consists of a generic implementation to handle both renewable and non-renewable resource types, as well as the release dates of projects. In addition, SGS can only produce a schedule if the order of jobs in the sequence respects all the precedence constraints and the selected modes satisfy that the total demand of each non-renewable resource does not exceed the availability. A sequence meeting such constraints is said to be *feasible*. Considering the infeasible ones, repair procedures are required.

2.1 Repair procedure

To repair violations of precedence constraints, each job, if required, is repositioned after its last predecessor in the sequence. For violated resources, the following procedure is applied. We compute the exceeding demands on non-renewable resources, then iterate over alternative modes of each job until it is possible to switch to another mode that strictly reduces the exceeding amounts. This iterative procedure stops as soon as the sum of the current demands does not exceed the availabilities any more, or until no alternative mode can be found. Considering the latter case, this procedure can fail and raises non-productive phases of the algorithm, but repairing from random non-feasible modes is a core component of the general scheme.

2.2 General algorithm scheme

Our algorithm is inspired from the $(\mu + \lambda)$ EA scheme [2], where individuals are sequences and their scores/performances are evaluated with SGS. The algorithm is initialized with μ randomly generated sequences. At each iteration, λ offspring sequences are generated from the current population using the crossover operator. Then the $\mu+\lambda$ sequences are sorted according to their scores so that individuals with the same performance and same start dates of jobs are forbidden, e.g. offspring sequences will replace parent ones. Finally, the μ best sequences among the $\mu + \lambda$ are kept as the population of the next iteration.

2.3 Crossover operator

The crossover operator is described as follows. Two parent sequences are first selected from the μ-population, and then a crossover point is randomly generated. The produced sequence will contain the mode keys before the crossover point of one parent joining with the mode keys after the crossover point of the other parent such that there is no duplicated jobs. However, this reconstruction can introduce missing jobs, e.g. jobs that are not presented in the parent parts. At this point of the algorithm, an intensive use of repairing procedures is made to greedily reintegrate missing jobs with randomly selected modes. Due to the partially random and modifying nature of this crossover operator, we considered that there is no need to add mutation operators.

For large instances, we also found out that a post-processing can improve the outcome solution of our EA. In next section, dedicated heuristics serving as post-processing of our algorithm are presented.

3 Post-processing with local search

The output sequence/solution of our EA is post-processed with an iterated local search procedure. In each iteration of this procedure, a random neighborhood is considered and the improvements are applied as soon as their detections. The local search is terminated when either no improvement can be found on all neighborhoods or the limit of runtime is reached. We use the four following neighborhoods.

3.1 TPD sort

The aim of this move is to reduce the TPD. Inspired from Jackson's rule [4], the idea is to first produce a sequence from applying an ascending stable sort algorithm on the current sequence with TPD as criteria. Then check if score of the produced sequence is better than the current one.

3.2 TPD join

This move has the same aim as the previous one. Given a sequence and its schedule, for each project, the latest jobs contiguously-scheduled are retrieved as a subsequence. Then we check if rejoining each of these subsequences with the remained jobs of the corresponding project can produce a new sequence so that the score is better than the current one.

3.3 Mode switch

In this move, all possibilities of switching a mode to another without altering the current schedule, i.e. the starting date of each job, is considered. Then we look for the mode switch that can improve the objectives TPD-TMS.

Data	Best result		Improvement
	TPD	TMS	of LS (in %)
A-1	1	23	0.00
A-2	2	41	0.00
A-3	0	50	0.00
A-4	67	43	0.00
A-5	169	109	0.00
A-6	168	108	0.00
A-7	633	194	0.00
A-8	324	150	0.00
A-9	247	132	0.00
A-10	1145	341	0.28
B-1	406	138	0.00
B-2	506	181	0.00
B-3	670	225	0.16
B-4	1486	290	0.02
B-5	1049	267	1.51
B-6	1283	278	3.85
B-7	935	241	0.37
B-8	3976	563	2.03
B-9	7183	1086	7.56
B-10	3650	505	3.15

Table 1: Performance of EA-LS on the public data sets.

3.4 Forward-Backward

The last improvement procedure is an adaptation of the Forward-Backward algorithm in scheduling. The adaptation mainly consists of using completion dates of projects resulting from forward phases to compute corresponding release dates of projects in the mirror problem as their difference with the total makespan. In addition, even when backward phases improve the objectives TPD-TMS, it can be necessary to shift all the start dates of the resulting jobs by a constant offset so that all release dates of projects (in the original problem) are respected.

To evaluate the sequences produced by TPD Sort, TPD Join, Mode Switch and Forward-Backward moves, SGS algorithm is used.

4 Experimental results

Our algorithms have been implemented in C++, using STL and boost libraries. To match the target machine of the challenge, the algorithms are also parallelized, thanks to the handy features of boost library. Our experiments were conducted on an Intel i7-960 3.2GHz. From those experiments, we found out that our LS is slow/costly when applied on random sequences of large instances. This justifies our choice of putting the LS in post-processing. We also decided to fix the parameters as follows: $\mu = 5$, $\lambda = 10$ and the last 5 seconds of the runtime is reserved for LS. Table 1 reports the results of our algorithm on the public data set A and B with 10 executions per instance. The first two columns show the best result for each instance in TPD-TMS. The other column indicates the average improvement of the post-processing procedure on the basis of the aggregated objective according to the *validator* program of [5].

To conclude, we have proposed a heuristic solution for MRCMPSP that makes use of an evolutionary algorithm and a local search. The algorithm requires a very few number of parameters. Our crossover operator appears to be a good and simple alternative method to handle the complex of both job sequencing and mode selection. Finally, due to the challenge deadline, we could not optimize our LS implementation so that the LS could be efficiently incorporated insides the EA instead of the post-processing phase. It seems that the proposed neighborhoods in the LS are promising for further research work on MRCMPSP with TPD as the main objective.

Acknowledgements We would like to thank to the IT service of Heudiasyc for the material assistance. We are also grateful to the organizers for giving us the opportunity to work altogether with such an interesting problem. This work was partially supported by the Regional Council of Picardie and the European Regional Development Fund (ERDF), under PRIMA project, and the Labex MS2T, which was funded by the French Government, through the program "Investments for the future" managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02).

References

1. C. Artigues, S. Demassey, E. Néron. Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. Wiley-ISTE, 2008.
2. T. Bäck. Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, USA, 1996.
3. M. R. Garey, D. S. Johnson. Computers and Intractability. A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.
4. L. A. Hall, D. B. Shmoys. Jackson's rule for single-machine scheduling: making a good heuristic better. Mathematics of Operations Research, vol 17(1), pp 22-35, 1992.
5. MISTA 2013 Challenge: http://allserv.kahosl.be/mista2013challenge/.

MISTA 2013

Combining Monte-Carlo and Hyper-heuristic methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem

Shahriar Asta · Daniel Karapetyan · Ahmed Kheiri · Ender Özcan · Andrew J. Parkes*

1 Introduction

This paper briefly describes our submission to the project scheduling challenge[1]. The full description of this problem domain can be found on the competition website and elsewhere; however, for completeness we briefly summarise it here. The broad aim is to schedule a set of different and partially interacting projects. Each project consists of a set of activities. The activities must respect a set of (hard) precedence constraints and project release times. Also the activities use resources and the appropriate resource limits are also hard constraints. There are basically two different ways to distinguish resources. Firstly, they can be either renewable or non-renewable. Renewable resources are ones that are available again at their full capacity whenever current activities stop using them, for example, they could be some machine. Non-renewable ones disappear on usage — an example could be fuel where one can take any amount of fuel, but only until the tank is empty. The second distinction between resources is that of 'local' resources that are associated with one of the projects, and 'global resources' that are shared between different projects. In the competition, there were no global non-renewable resources, and so the only interaction between projects is from the global renewable resource(s). A complication is that each activity actually can be performed using any one of a set of 'modes'. The mode determines the set of resources used by the activity and the duration of the activity (though note that the modes do not affect the set of precedences). A solution consists of an assignment of mode and starting time to every activity and that satisfies all the precedence and resource constraints.

Given a solution then each project i has an associated makespan MS_i which is the time from it being released to the time the last activity is completed. The primary objective is to minimise the "Total Project Delay" (TPD), which (up to constant terms) is the sum of the makespans MS_i for each project i. The tie-breaking secondary objective is to minimise the overall "Total Makespan" (TMS), which (up to a constant term) is the finishing time of the last activity.

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
{ sba, dxk, axk, exo, ajp }@cs.nott.ac.uk
∗ Contact Author

[1] http://allserv.kahosl.be/mista2013challenge/

In designing an algorithm, there are two 'natural' representations to be used in the search for an assignment of activities to times:

Schedule-based: A direct representation using the assignment times of activities.
Sequence-based: This is based on a selecting a total order on all the activities. Given such a sequence then a time schedule is constructed by taking the activities one at a time in the order of the sequence and placing each one at the earliest time that it will go in the schedule.

The schedule based method is direct, and (perhaps) most natural for a mathematical programming approach, but we believed it could make a metaheuristic method difficult, in particular, it becomes more challenging to generate many feasible solutions. Hence, we used the sequence method as it has the advantage of being easier to produce schedules that are both feasible and for which no activity in such a schedule can be moved to an earlier time without moving some other activities. Our overall approach is in two phases in a "construct and improve" fashion. Firstly, a heuristic constructor is used to create initial sequences. The novelty here is that we generate it using Monte-Carlo Tree Search (MCTS) methods [1]. Secondly, an improvement phase is applied using a large variety of heuristic neighbourhood moves. This phase is carefully controlled by a combination of methods arising from standard metaheuristics, memetic algorithms, and also an extension of an existing hyper-heuristic [2].

2 Construction Phase

On inspecting good solutions, we found many cases had an approximate ordering of the projects: There would be times in the schedule when the general focus would be on one or few projects and during the schedule this focus would change. This is natural when the primary objective is the delay-based TPD rather than on the overall makespan, and it suggested that a good constructor should attempt to create initial sequences that mimic such project (partial) orderings. The problem then is how to quickly compare between different project orderings. We use an MCTS method in which the fast "rollout" is done by means of a randomised schedule constructor. As standard in MCTS, the aim is not to directly produce good solutions but to use an unbiased sampling to make good decisions. There is also an option as to whether to generate total or partial orders between the projects. We settled for a 3-way partition of the projects taken to correspond to 'start', 'middle' and 'end' of the overall project time. The MCTS is then used to select the projects constituting each of the partitions. In the rollout, the order of activities within each project partition is taken as a random one that respects the precedences.

3 Improvement Phase

The improvement phase uses an extension of [2] together with a large set of moves, or "Low Level Heuristics". Some of which are given below; all moves are restricted, as needed, to only generate sequences that respect all the precedence constraints.

1. SwapJobs: Swap two activities within the sequence.
2. InsertJob: Insert a given activity into a new location in the solution sequence.
3. SetMode: Change the mode of a activity to a new randomly chosen mode.

4. MoveUniform: A set of activities is selected uniformly at random. The move has three options: moving activities, changing modes or both. When moving the selected activities they are reshuffled randomly. When changing the modes, random modes are assigned to the selected activities.
5. MoveLocal: Similar to MoveUniform, with the difference being that the activities are selected using a specific distribution that in not uniform but that is centered around a controllable position, and with a controllable width, within the sequence.
6. MoveOneProject: Similar to MoveUniform, but the activity selection is biased towards a randomly selected project.
7. MoveBiasedGlobalResource: Similar to MoveUniform, but favours selection of activities which have a bigger remaining capacity on the global resources.
8. MoveEndBiased: Similar to MoveBiasedGlobalResource, but favours the activities with a position close to the end of their project.
9. FILS swapJobs/insertJobs/setMode: these three moves are First Improvement Local Search (FILS) procedures based on the swapping, inserting or mode changing limited neighbourhoods.
10. SwapTwoProjects: Swaps two randomly selected projects in the sequence.
11. MutationOneExtreme: the activities of a randomly selected project are all collected and squeezed into a randomly-selected position in the sequence.
12. MutationOne: Shifts all the activities of a randomly selected project by a fixed number of positions in the sequence.
13. MoveProjects: Extracts the sequence of the projects in the solution (based on the positions of the last activities in each project), selects several consequent projects, and then moves them to either the beginning or end of the sequence.

The moves are controlled by a combination of meta-heuristics and a hyper-heuristic, all in the context of a multi-threaded population-based approach that uses ideas from memetic algorithms.

4 Implementation Experiments and Results

As might be expected, the many of our experiments were concerned with selecting the algorithms and then (partially) tuning the many parameters that are possible within the overall search control algorithm. It was also important to a careful implementation so that the construction of the schedule from a sequence was as fast as possible. For example, it helped to use "prefix sharing" in which the construction from the current sequence can reuse the results of a previous construction, at least up until the point at which the two activity sequences differ.

We provide some preliminary experimental results in Table 1. For reference purposes, we include the results from the competition website of the best results of the qualification phase. We then give our average performance of the submitted code under conditions similar to the competition, and, for purposes of comparison, the best values that we ever encountered during all our experiments. It can be seen that under competition conditions, we were generally able to beat (or equal) the results from the challenge website, but that there is still room for improvement on most of the instances.

Instance	Quals best		Our Mean		Our Best	
A-1	1	23	1	23	1	23
A-2	2	41	2	41	2	41
A-3	0	50	0	50	0	50
A-4	65	42	65	42	65	42
A-5	153	105	152	104	150	104
A-6	147	96	136	91	133	91
A-7	596	196	602	202	589	201
A-8	302	155	280	151	272	151
A-9	223	119	203	126	196	125
A-10	969	314	850	308	840	306
B-1	—	—	352	128	345	125
B-2	—	—	441	169	424	165
B-3	—	—	547	211	526	208
B-4	—	—	1279	285	1254	277
B-5	—	—	831	247	809	251
B-6	—	—	936	233	897	224
B-7	—	—	796	231	782	229
B-8	—	—	3315	534	3088	533
B-9	—	—	4197	755	4097	745
B-10	—	—	3238	454	3136	445

Table 1 Experimental results summary. The objective values are given as ordered pairs of "TPD", total project delay, and "TMS", total makespan. The 'Quals best' values are from the competition website for all the entrants in the qualification round. 'Our Mean' values are average values for our algorithm under time and machine conditions intended to match those of the competition. 'Our Best' is the best solution encountered over a large number of runs with a variety of runtimes and parameter settings.

5 Conclusions

Our algorithm consists of a two-phase construct-and-improve method working on the sequence in which activities are given to a schedule constructor. The construction of an initial activity sequence is done by a (novel) hybrid of MCTS and partitioning of the projects. The improvement phase uses a large number of neighbourhood moves, in a multi-threaded fashion, and controlled by a mix of ideas from metaheuristics, memetic algorithms, and hyper-heuristics.

References

1. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. Computational Intelligence and AI in Games, IEEE Transactions on **4**(1), 1–43 (2012). DOI 10.1109/TCIAIG.2012.2186810
2. Kheiri, A., Özcan, E., Parkes, A.J.: HySST: Hyper-heuristic search strategies and timetabling. In: Proceedings of the Ninth International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012) (2012)

MISTA 2013

An Integer Programming Approach for the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

Túlio A.M. Toffolo · Haroldo G. Santos · Marco A.M. Carvalho · Janniele A. Soares

1 Introduction

This paper briefly describes our proposed approach for the MISTA 2013 Challenge (MISTA Conference, 2013) for the multi-mode resource-constrained multi-project scheduling problem (MRCMPSP). Specific details about the project scheduling problems can be found on Demeulemeester and Herroelen (2002).

We produced a hybrid algorithm with several Integer Programming (IP) based components:

- priority based fixed mode multi-start constructive algorithm;
- IP constructive algorithm;
- IP local search algorithm;
- forward-backward improvement (FBI) procedure.

These components are detailed in the following sections.

2 Initial Solution

The major obstacle for the production of feasible solutions using greedy algorithms for the MRCMPSP are the non-renewable resources. While the use of renewable resources only impacts the delay/speedup of the projects, the use of non-renewable resources (NRN) can easily produce infeasible solutions. One option is the definition of job modes beforehand so that they respect the use of NRN. We denote the problem of selecting these initial modes as *ModeMin*. In this problem, we want to define, for each job, a mode such that the total non-renewable resources consumption limits are always respected and the duration of jobs is minimized. *ModeMin* itself is as hard as the m-dimensional knapsack problem, so we solve a small IP to obtain these modes.

The *ModeMin* solution gives a valid estimative of NRN consumption for non-allocated jobs and guides the greediness level of our constructive schedule generation algorithms.

Túlio A.M. Toffolo, Haroldo G. Santos, Marco A.M. Carvalho, Janniele A. Soares
Computing Department, Federal University of Ouro Preto
E-mail: tulio@toffolo.com.br, haroldo@iceb.ufop.br, jannysdp@hotmail.com, marco.opt@gmail.com

During the schedule generation, if a job violates any non-renewable resource consumption limit, then its mode is changed to the defined *ModeMin*.

A heuristic IP decomposition is used to build an initial feasible solution, as follows. At each iteration, a time window is defined starting at time zero and an IP model is created and solved with the objective of allocating the maximum number of jobs within this window. The model considers the original constraints, such as precedence and resource consumption. These time windows are sequential and non-overlapping. The algorithm finishes when all jobs are allocated.

The pseudocode for generating the initial solution is shown in Algorithm 1. The algorithm takes as input data: (i) all the jobs that make the projects, (ii) the window size, (iii) the time limit, (iv) a two-dimensional vector with the earliest start time (EST) of all jobs in each mode and (v) a vector containing the amount of non-renewable resources that will be needed to complete the project. This last vector is based on the alocation of the jobs on *ModeMin*. At each iteration the algorithm selects jobs to be allocated, taking into account the jobs that have the EST inside or before the time window. After this selection, an IP model is created considering the selected jobs, as detailed in section 4.3. The next step is to update the vector of future resources and the vector of EST by taking into account the new allocation of the jobs. After that, the time window moves forward. The output of the algorithm is a feasible initial solution.

Algorithm 1: Heuristic Constructive

```
Data: windowSize, timeLimit, J, EST[][], resourceFuture[]
Result: solution
minTime ← 0; a ← 0;
while (solution is not valid and time limit not reached) do
    J' ← updateAvailableJobs(J, EST) ;
    createModel(J', minTime, minTime + windowSize);
    optimizeModel();
    updateSolution(solution);
    updateResourceFuture(resourceFuture);
    updateEST(J', EST);
    minTime ← minTime + windowSize;
```

It is important to note that defining the size of the window time itself is tricky problem. The time window must be small enough to ensure that the generated IP models are easily solved and, at the same time, must be big enough to indicate relevant allocations.

3 Local Search

We also employ an IP-based local search procedure which has similarities with the previously described algorithm. A window is defined and only jobs within this window are considered eligible to have their modes and starting times changed.

The IP based local search is combined with the forward-backward improvement (FBI) procedure, also called double justification procedures (Vallsa et al, 2005). This is a two step improvement phase to a single project schedule generated by some other method. The method is outlined in the following paragraphs.

In the first step (the forward pass), the jobs are right justified in the schedule, that is, except for the first and the last jobs, all job allocations are shifted to the right, starting from the final job immediate predecessors until the initial job immediate successors. This step generates a right active schedule – a schedule where no activity can be finished later without advancing some other activity or increasing the makespan (Vallsa et al, 2005). Since the final job is not shifted, the current makespan is held. If during the first step a slack in the schedule is generated, the second step tries to reduce the makespan by eliminating that slack. In this second step (the backward pass), jobs are analogically left justified in the schedule, that is, except for the initial job, all job allocations are shifted to the left, starting from the initial's immediate successors until the final job. This step generates a left active schedule (a schedule where no activity can be started earlier without delaying some other activity – or violating the constraints).

In both steps the schedule is always feasible, since the precedence relations and resource consumption constraints are satisfied. Each shift is performed using the Serial Generation Scheme.

After the application of the FBI, some jobs may not have been shifted in any direction and may represent the problem's bottleneck. We propose a third improvement step that marks these stationary jobs and then randomly changes each marked job's mode and tries to perform a left-shift using the Serial Generation Scheme.

These three steps are performed while there is an improvement in the quality of the solution and until the time limit is not reached.

The pseudo-code of the local search is shown in Algorithm 2. The algorithm takes as input data: (i) an initial solution, (ii) the time window size, (iii) the time limit, (iv) the number maximum iterations to make changes in a solution, (v) the number of iterations of changes in a sequence of allocation, (vi) the maximum of global iterations, and (vii) the maximum, minimum and actual size of the perturbations. Until the time limit is not reached, the algorithm applies FBI while the solution is being improved. After that, the modes are changed using the IP model. Changes in the solution are made through a sequence of allocations until the maximum number of iterations is reached. Afterwards, when no improvement is observed, another new solution is obtained. We basically use the procedure shown in section 1, but we use the priority observed in the best solution to guide the generation of the new solution. The output of the method is a possibly improved feasible solution.

Algorithm 2: Heuristic Local Search

```
Data: solution, windowSize, timeLimit, maxIters, itersSwapMove, maxGlobalIters,
      maxPerturbations, minPerturbations, realPerturbation
Result: solution
perturbationSize ← minPerturbations;
iters ← 0; globalIters ← 0;
minTime ← 0 ; maxTime ← 0;
bestSolution ← cloneSolution(solution);
while ( time limit not reached ) do
    while ( ++iters < maxIters ) do
        fo ← f(solution);
        biasedForwardImprovement(solution);
        biasedBackwardImprovement(solution);
        curSol ← f(solution) ;
        stationaryJobs ← NULL ;
        while ( curSol > f(solution) and stationaryJobs is not empty ) do
            curSol ← f(solution);
            modeChangeStationaryJobs(solution, stationaryJobs);
            biasedBackwardImprovementStationaryJobs(solution, instance,
            stationaryJobs);
            biasedBackwardImprovement(solution, stationaryJobs);
        while ( swapMoveJobs(solution,itersSwapMove)) do
        if ( f(solution) < f(bestSolution) ) then
            iters ← 0;
            globalIters ← 0;
            perturbationSize ← minPerturbations;
            bestSolution ← cloneSolution(solution);
        else
            if ( f(solution) < fo ) then
                iters ← 0;
                perturbationSize ← minPerturbations;
    if (++globalIters ≥ maxGlobalIters) then
        perturbationSize ← (perturbationSize + 1) % maxPerturbations;
        globalIters ← 0;
    if ( perturbationSize = maxPerturbations - 1 and realPerturbation > 0) then
        perturbationSize ← minPerturbations;
        guidedRestart(solution);
    else
        for ( p ← 0; p < perturbationSize ; p++ ) do
            minTime ← rand() % maxTimeSlot;
            maxTime ← minTime + (windowSize * (double)(50 + rand() % 100)) /
            100;
            createModelSubProblem(minTime, maxTime);
            optimizeModel();
            updateSolution();
```

4 IP Models

The models presented in this section are used by the algorithms presented in sections 2 and 3.

4.1 Input Data

The following input data are defined for the three models presented on the following sections. These models were inspired in the models proposed by Kolisch and Sprecher (1996).

- J: set of jobs;
- P: set of projects;
- K: set of non-renewable resource;
- R: set of renewable resource;
- T: maximum time;
- C: set of paths;
- j: job j;
- m: mode m;
- p: project p;
- k: non-renewable resource k;
- r: renewable resource r;
- t: time t;
- c: path c;
- d_{jm}: duration of the job j on mode m;
- q_{kjm}: needed amount of non-renewable resource k to perform job j on mode m;
- q_{rjm}: needed amount of renewable resource r to perform job j on mode m;
- q_k^{disp}: available quantity of the non-renewable resource k;
- q_r^{disp}: available quantity of the renewable resource r;
- EST_{jm}: earliest start time of the job j on mode m;
- LFT_{jm}: latest finish time of the job j on mode m;
- usK_k: quantity of the non-renewable resource k used;
- usR_{tr}: quantity of the renewable resource r used at time t;
- fuK_k: amount needed for future non-renewable resource k;
- cP: constant weight of the higher value between the predecessor which ends later or duration of the predecessor analysed;
- cW: constant weight that multiplies 10 by the last time analyzed incremented by 1 plus the greater distance from the job until the final task of the project, multiplied by the ratio of the minimum distance of the job until the final task of the project and the duration of the projects more the duration of the job on mode min;
- cM: constant weight with a random value high for modes that are equal to the mode in which the activity is allocated.

4.2 IP *ModeMin*

The objective functions (1) and (2) are responsible for minimizing the duration of the jobs and projects, respectively. The constraint (3) is responsible for ensuring that a job is allocated only once. Constraint (4) ensures that there will be extrapolated amounts of non-renewable resources. Constraint (5) is responsible for restricting the decision variable z.

The following defines the decision variables for the model *ModeMin*:

- x_{jm}: binary variable: 1 if the job j has already been performed on mode m, 0 otherwise;

– z_p: integer variable: defines the end of the project.

$$\textbf{Minimize}$$
$$\sum_{j\in J}\sum_{m\in M}(d_{jm}x_{jm}) \quad (1)$$
$$\sum_{p\in P} z_p \quad (2)$$

$$\textbf{Subject to}$$
$$\sum_{m\in M} x_{jm} = 1 \qquad \forall j \in J \quad (3)$$
$$\sum_{j\in J}\sum_{m\in M}(q_{kjm}x_{jm}) \leq q_k^{disp} \qquad \forall k \in K \quad (4)$$
$$\sum_{c\in C}\sum_{j\in J}\sum_{m\in M}(d_{jm}x_{jm}) \leq z_p \qquad \forall p \in P \quad (5)$$

4.3 IP Constructive

The objective functions (6), (7) and (8) are responsible for maximizing the number of jobs assigned, minimizing the end of each project and minimizing the overall finish, respectively. Constraint (9) ensures that the allocations are unique for the job but allows a job not to be allocated. Constraints (10) and (11), respectively, ensure that the amount of non-renewable and renewable resources are not extrapolated. Constraints (12) and (13) ensure the precedence relationship. Finally, constraints (14) and (15) are responsible for the activation of the decision variables y and $yMax$.

The following defines the decision variables for the model Constructive:

– x_{jmt}: binary variable: 1 if the job j has already been performed on mode m and started at t, 0 otherwise;
– y_p: integer variable: defines the end of the project;
– $yMax$: integer variable: defines the highest order among the projects.

$$\textbf{Maximize}$$
$$\sum_{j\in J}\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt} \quad (6)$$
$$-\sum_{p\in P} y_p \quad (7)$$
$$-yMax \quad (8)$$

$$\textbf{Subject to}$$
$$\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt} \leq 1 \qquad \forall j \in J \quad (9)$$

$$\sum_{j\in J}\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}}(q_{kjm}\, x_{jmt}) + \sum_{j\in J}\left(\left(1-\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt}\right) q_{kj\tilde{m}_j}\right) \leq q_k^{disp} - usK_k - fuK_k \qquad \forall k \in K \quad (10)$$

$$\sum_{j\in J}\sum_{m\in M}\sum_{q=max(t-d_{jm}+1,EST_{jm})}^{min(t,LFT_{jm})}(q_{rjm}\, x_{jmq}) \leq q_r^{disp} - usR_{tr} \qquad \forall r \in R, \forall t \in T \quad (11)$$

$$\sum_{m\in M}\sum_{t=EST_{j_1m}}^{LFT_{j_1m}}((t+d_{j_1m})\, x_{j_1mt}) - \sum_{m\in M}\sum_{t=EST_{j_2m}}^{LFT_{j_2m}}(t\, x_{j_2mt}) - \left(1-\sum_{m\in M}\sum_{t=EST_{j_2m}}^{LFT_{j_2m}} x_{j_2mt}\right) cP \leq 0 \qquad \forall (j_1,j_2) \in Pred \quad (12)$$

$$\sum_{m\in M}\sum_{t=EST_{j_1m}}^{LFT_{j_1m}} x_{j_1mt} - \sum_{m\in M}\sum_{t=EST_{j_2m}}^{LFT_{j_2m}} x_{j_2mt} \geq 0 \qquad \forall (j_1,j_2) \in Pred \quad (13)$$

$$\left(1-\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt}\right) cW \leq y_p \qquad \forall p \in P, \forall j \in J \quad (14)$$
$$yMax - y_p \geq 0 \qquad \forall p \in P \quad (15)$$

4.4 IP Change Modes

The objective function (16) is responsible for maximizing the variation on modes of the jobs. The constraint (17) is responsible for ensuring that a job is allocated only once. Constraints (18) and (19) respectively ensure that the amount of non-renewable and renewable resources used do not extrapolate the limits imposed. Constraint (20) ensures the precedence relationship.

The following defines the decision variables for the mode IP Change Modes:

– x_{jmt}: binary variable: 1 if the job j has already been performed in the mode m and started at t, 0 otherwise.

Maximize

$$\sum_{j\in J}\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} cM\ x_{jmt} \tag{16}$$

Subject to

$$\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} x_{jmt} = 1 \qquad \forall j \in J \tag{17}$$

$$\sum_{j\in J}\sum_{m\in M}\sum_{t=EST_{jm}}^{LFT_{jm}} \left(q_{kjm}\ x_{jmt}\right) \leq q_k^{disp} - usK_k \qquad \forall k \in K \tag{18}$$

$$\sum_{j\in J}\sum_{m\in M} \sum_{q=max(t-d_{jm}+1,EST_{jm})}^{min(t,LFT_{jm})} \left(q_{rjm}\ x_{jmq}\right) \leq q_r^{disp} - usR_{tr} \qquad \forall r \in R, \forall t \in T \tag{19}$$

$$\sum_{m\in M}\sum_{t=EST_{j_1m}}^{LFT_{j_1m}} \left((t+d_{j_1m})\ x_{j_1mt}\right) - \sum_{m\in M}\sum_{t=EST_{j_2m}}^{LFT_{j_2m}} t\ x_{j_2mt} \leq 0 \qquad \forall (j_1,j_2) \in Pred \tag{20}$$

5 Computational Experiments

All algorithms were coded in C++ and the IP models were solved by Gurobi 5.5. The code was compiled with GCC 4.6.2 using flag -O3. The competition is current ongoing. In the qualification phase we were ranked in second place among 16 teams with this approach.

The code was parallelized in 3 threads and in each one the parameters were set to different values. These values were obtained after several empirical tests.

In the first thread, in the construction of the initial solution the parameter windowSize was set to 40 and the local search parameters were set such that: itersSwapMove=10, maxIters=10, maxGlobalIters=10, maxPerturbations=15, realPerturbation=1 and windowSize=10.

In the second thread, the parameter windowSize in the initial solution generator was also set to 40 and the local search parameters were set such that: itersSwapMove=1, maxIters=1, maxGlobalIters=20, maxPerturbations=10, realPerturbation=1 and windowSize=10.

In the third thread, in the construction of the initial solution the parameter windowSize was set to 20 and the local search parameters were set such that: itersSwapMove=5, maxIters=1, maxGlobalIters=20, maxPerturbations=10, realPerturbation=0 and windowSize=10.

Table 1 shows the results obtained with the approach using a computer similar to the one used by the competition organizers (MISTA Conference, 2013).

Table 1 Best and average results after 50 runs of the algorithm with different random seeds in 300 sec

Instance	Best		Average	
	TPD	**TMS**	**TPD**	**TMS**
A-1	1	23	1	23
A-2	2	41	2	41
A-3	0	50	0	50
A-4	68	46	68	46
A-5	159	103	191	106
A-6	170	93	189	102
A-7	639	193	771	212
A-8	281	145	348	147
A-9	212	122	269	126
A-10	983	303	1182	332
B-1	360	131	360	131
B-2	431	155	437	160
B-3	585	199	619	207
B-4	1435	290	1504	291
B-5	867	250	938	251
B-6	970	224	1157	233
B-7	902	239	911	245
B-8	3001	521	3244	533
B-9	4807	746	5339	769
B-10	3123	425	3198	437

Acknowledgements The authors thank CNPq and FAPEMIG for supporting this research.

References

Demeulemeester EL, Herroelen WS (2002) PROJECT SCHEDULING: A Research Handbook. Kluwer Academic Publishers, Leuven Belgium

Kolisch R, Sprecher A (1996) PSPLIB - a project scheduling problem library. European Journal of Operational Research 96:205–216

MISTA Conference (2013) Mista 2013 challenge. URL `http://allserv.kahosl.be/mista2013challenge/`, access date: 2 aug. 2013

Vallsa V, Ballestina F, Quintanillab S (2005) Justification and RCPSP: A technique that pays. European Journal of Operational Research 165(2):375–386

Plenary Papers

LP, or not LP: that is the question

Federico Della Croce
D.A.I. Politecnico di Torino, Italy
CNR, IEIIT, Torino, Italy
federico.dellacroce@polito.it

Abstract

LP, or not LP: that is the question.
Whether 'tis nobler in the mind to suffer
The slings and arrows of linear programming,
Or to take arms against a sea of metaheuristics,
And, by opposing, end them? CPLEX, XPRESS; and more.
It is well known that in the 20th century, linear programming (LP) modeling, even though strongly applied to combinatorial optimization, was not too successful when directed to scheduling problems. Purpose of this talk is to show that the field of successful applications of LP (and MP - mathematical programming) modeling is still growing and includes also scheduling topics. We first focus on the single machine total completion time scheduling problem with different release times and show that the related positional completion times LP formulation embedding cuts derived from the combinatorial structure of the problem performs reasonably well when handled by means of an ILP solver. Then, we show how LP can be used to tighten bounds for approximation results in scheduling theory. Similarly, we show how to enhance the complexity bounds of branch-and-reduce exact exponential algorithms by means of the so-called measure-and-conquer paradigm requiring always the solution of a specific MP model. Finally, we present a basic constructive LP-based approach to generate proper logic puzzles such as SUDOKU and FUTOSHIKI containing always a unique solution.

1 Introduction

A combinatorial optimization problem can be typically formulated as a mathematical programming (MP) by first translating each elementary decision into a variable and by then specifying the objective function and the related constraints. To obtain a linear programming (LP) model, the objective is also required to be a linear function of the variables and the constraints are required to be linear (in)equalities involving the variables. We refer to Integer Linear Programming (ILP) if the variable are also supposed to be integer. While modeling can sometimes be considered to be an art which cannot be reduced to a standard set of procedures, there exist however references (see for instance [16, 21]) that discuss some sort of systematic approach in the MILP modeling of combinatorial optimization problems. The purpose of this work is actually to indicate that LP and MP modeling is still a strong tool in combinatorial

optimization and is more and more applicable to topics not yet considered in the 20th century. To this extent, we show in Section 2 that pure sequencing problems can be efficiently solved by means of an ILP solver if a proper formulation exploiting the problem properties is used. Also, we show in Section 3 that approximation results can be derived by means of the iterated solution of several ILP models. Then, we deal in Section 4 with exact exponential branch-and-reduce algorithms for combinatorial problems and show that the so-called *measure-and-conquer* paradigm requires the solution of a MP model to improve the worst-case complexity bound of the exact algorithm. Finally, in Section 5 we consider well known logic puzzles such as Sudoku and Futoshiki and show that the generation of proper puzzles, that is puzzles having a unique solution, can easily obtained by applying a simple constructive approach based on the repated solutions of several ILP models.

2 Positional completion times ILP formulation in single machine scheduling

ILP based formulations in single machine scheduling were already successful when dealing with the static single machine problem with the objective of minimizing the weighted sum of tardy jobs ($1||\sum w_jU_j$) [15, 2] where, particularly, a dedicated exact algorithm proposed in [2] exploiting the ILP formulation first introduced in [15] was able to solve problems with up to 50000 jobs in size. However, in [15] it was shown that the $1||\sum w_jU_j$ problem can be modeled as a multidimensional 0/1 knapsack problem with a 0/1 variable x_j for each job j where $x_j = 1$ if j is early (viceversa $x_j = 0$). That formulation was also generalized to the case with deadlines ($1|\overline{d_j}|\sum w_jU_j$) in [2] where problems with up to 30000 jobs in size were solved to optimality. However in both cases, the problems were not sequencing problems but rather assignment problems as the only decision required was to define for each job j wether j was early or tardy, the sequence being univocally determined once the assignment was defined. Here, we consider the single machine total completion time sequencing problem subject to release times ($1|r_j|\sum C_j$) where also sequencing decisions must be taken and that can be stated as follows. A set N of n jobs must be processed on a single machine which is always available. Each job j has a processing time p_j and a release time r_j. The machine can process one job at a time and the processing of each job j cannot start before r_j. Preemption is not allowed. The objective is to minimize the sum of completion times $\sum C_j$ where C_j denotes the completion time of job j. The problem is known to be $\mathcal{NP}$-hard in the strong sense. We assume that the jobs are re-indexed in such a way that $p_1 \leq p_2 \leq ... \leq p_n$ where in case of ties ($p_i = p_{i+1}$) $r_i \leq r_{i+1}$. Notice that, if preemption is allowed, the related $1|r_j, pmtn|\sum C_j$ problem is solvable in $O(n \log n)$ time by means of the so-called SRPT (Shortest Remaining Processing Time) priority rule ([1]), where at any time the job with shortest remaining processing time among the available jobs is scheduled.

2.1 Problem formulation and valid inequalities

Several ILP formulations can be derived for the $1|r_j|\sum C_j$ problem. In [7], the positional variables ILP formulation (see [13]) was considered that turned out

to be the best one with respect to other classical formulations. That formulation is depicted below.
Let $C_{[j]}$ be the completion time of the j-th job processed and let x_{ij} be a 0/1 variable, where $i, j \in \{1, \ldots, n\}$. A variable x_{ij} is equal to 1 if job i is in position j, and 0 otherwise. The ILP model is as follows.

$$\min \sum_{j=1}^{n} C_{[j]} \tag{1}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j = 1, \ldots, n \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1, \ldots, n \tag{3}$$

$$C_{[1]} = \sum_{i=1}^{n} (p_i + r_i) x_{i1} \tag{4}$$

$$C_{[j]} \geq C_{[j-1]} + \sum_{i=1}^{n} p_i x_{ij} \qquad \forall j = 2, \ldots, n \tag{5}$$

$$C_{[j]} \geq \sum_{i=1}^{n} (p_i + r_i) x_{ij} \qquad \forall j = 2, \ldots, n \tag{6}$$

$$x_{ij} \in \{0, 1\}, \quad C_{[j]} \geq 0 \tag{7}$$

where constraints (2)–(3) state that a job is chosen for each position in the sequence and each job is processed exactly once. Constraints (4) set the completion time of the first job. Constraints (5)–(6) forbid for each job to start before the job in the previous position completes and before its release date.

The above formulation can be strenghtened by adding valid inequalities derived from properties proposed in [4] and [8]. We make use of the following notation. Let S denote the SRPT schedule and let OPT denote the optimal sequence.

In [4], it was shown that, for all position j, the completion time of the j-th job in OPT is greater than or equal to the completion time of the j-th job in S, that is $C_{[j]}(OPT) \geq C_{[j]}(S)$.

In [8], an improvement over the SRPT lower bound running in $O(n^2)$ time has been proposed that applied a logical reasoning on positions in an optimal schedule. We sketch here how this improvement is computed. We know that a job k completed in position $[j]$ in S will occupy some position in the optimal sequence OPT and this issue can be formalized according to the following three exhaustive cases: job k occupies position $l \geq [j+1]$, job k occupies position $h \leq [j-1]$, or job k occupies position $[j]$. In [8] it is shown that, as one of the above cases must always hold, it is possible to compute in $O(n^2)$ time a coefficient θ_j (for all positions j) indicating a lower bound on the increase in OPT of the completion times of the jobs placed in positions [j 1], [j] and [j + 1] with respect to the jobs occupying the same positions in the $SRPT$ sequence S, that is $C_{[j-1]} + C_{[j]} + C_{[j+1]} \geq C_{[j-1]}(S) + C_{[j]}(S) + C_{[j+1]}(S) + \theta_j$. Notice

that the position $j-1$ makes no sense for $j=1$ and position $j+1$ makes no sense for $j=n$.

Combining the above results, the following property was proved in [7].

Property 1. *The following are valid inequalities that can be added to the problem model.*

$$C_{[j]}(OPT) \geq C_{[j]}(S) \quad \forall j \in 1,...n \tag{8}$$

$$C_{[1]} + C_{[2]} \geq C_{[1]}(S) + C_{[2]}(S) + \theta_1 \tag{9}$$

$$C_{[j-1]} + C_{[j]} + C_{[j+1]} \geq C_{[j-1]}(S) + C_{[j]}(S) + C_{[j+1]}(S) + \theta_j \quad \forall\ j = 2 \dots n-1 \tag{10}$$

$$C_{[n-1]} + C_{[n]} \geq C_{[n-1]}(S) + C_{[n]}(S) + \theta_n \tag{11}$$

2.2 Computational experimentation

A computational experimentation was proposed in [7] on a QuadCore 2.67 GHz PC with 1GB RAM to evaluate the impact of the above valid inequalities when solving the ILP formulation by CPLEX 12.3 solver. In Table 1, we report the benchmark on solution quality and time of the pure model and the enhanced one over the first ten testbed instances with 50 jobs taken from [12] and a timelimit of 100 seconds. As we can see, the pure model is able to reach the optimal solution within the time limit in just 7 cases over 10 and to prove optimality in just 4 cases. Viceversa, the enhanced model is able to solve optimally all instances in few seconds.

We summarize hereafter the other findings of [7]. When all 200 testbed instances with size $n = 50$ of [12] are considered, CPLEX 12.3 solver applied to the enhanced model requires on the average 2.5 seconds and in the worst case 25.5 seconds. When we consider the testbed instances with size $n = 100$, CPLEX 12.3 solver applied to the enhanced model requires on the average 93 seconds and in the worst case 2685 seconds. Notice that the state of the art exact algorithms proposed in [20] and [18] still outperform the ILP based approach, that however becomes comparable if problems with larger distribution of CPU times (processing times drawn at random from discrete uniform distribution $[1-1000]$) are considered. Further, when the enhanced formulation is applied within a matheuristic approach, the proposed heuristic algorithm strongly dominates the state of the art heuristic approaches being able to find with reasonable CPU times near optimal solutions for problems with up to 500 jobs in size.

3 Computing bounds for approximation results in scheduling

A wide area of research relates to the search for approximation algorithms for $NP-hard$ scheduling problems. Then, whenever polynomial time approximation schemes are not reachable, it is common to search for polynomial time algorithms capable of reaching constant approximation ratios. If a minimization problem is considered, where $f(OPT)$ is the optimal solution value, then we look for a polynomial algorithm A capable of reaching a heuristic solution H such that the ratio $\rho = \frac{f(H)}{f(OPT)}$ is not superior to some given constant k. To prove this, we need to show that, for any instance, algorithm A reaches a

Instance name	Size (jobs)	optimal sol. value	Pure Model sol. value	Time sec.	Enhanced Model sol. value	Time sec.
P50_01	50	**40490**	**40490**	≥ 100	**40490**	1.66
P50_02	50	**43845**	**43845**	37.19	**43845**	1.72
P50_03	50	**46047**	**46047**	7.25	**46047**	2.53
P50_04	50	**49004**	**49004**	27.28	**49004**	0.39
P50_05	50	**50653**	**50653**	4.78	**50653**	0.48
P50_06	50	**56308**	56309	≥ 100	**56308**	2.61
P50_07	50	**54966**	54979	≥ 100	**54966**	0.63
P50_08	50	**52402**	**52402**	≥ 100	**52402**	5.63
P50_09	50	**50248**	50322	≥ 100	**50248**	2.13
P50_10	50	**47313**	**47313**	≥ 100	**47313**	2.24

Table 1: Impact of the valid inequalities on ten 50 jobs testbed instances

solution such that $\rho > k$ can never occur. Then we say that such ratio is *tight* if we are able to find an instance of the problem such that algorithm A performs with ratio $\rho = k$, or it is *asymptotically tight* if we are able to find an instance of the problem such that algorithm A performs with ratio ρ that tends asymptotically to k. Else, we search for an instance which is close enough to the upper bound k previously computed. The search for such an instance is indeed doable by means of LP modeling techniques. In this section, we show, as an example, that LP modeling can be successfully applied to find a lower bound to the approximation ratio in a two-machine flowshop scheduling problem under the re-optimization paradigm.

3.1 Approximating the 2-machine total competion time flowshop problem in the re-optimization setting under job insertion

Consider the solution of machine scheduling problems in the re-optimization setting, which can be described as follows: considering an instance I of a given problem Π for which an optimal solution OPT is provided, and an instance I' which results from a local perturbation of I, can the information provided by OPT be used to solve I' in a more efficient way (i.e., with a lower complexity and/or with a better approximation ratio) than if this information wasn't available? This is also related to the so-called *reactive scheduling* (see [17]), that is what refers to the schedule modifications that may have to be made during project execution starting from a baseline schedule. In some cases, the reactive scheduling effort may rely on very simple techniques aimed at a quick schedule consistency restoration. To this purpose, by considering an optimal schedule as baseline schedule and some jobs additions or deletions as schedule disruptions, we can see that scheduling re-optimization may be considered as strictly linked to reactive scheduling, particularly with simple reoptimization strategies so that the baseline schedule is only mildly modified.

Here, we study as a simple example the 2-machine total completion time flowshop problem ($F2 \mid\mid \sum C_j$ according to the three-field notation of [14]) that can be stated as follows. A set of n jobs is available at time 0 to be processed on 2 machines where each job is processed on machine M_1 first and then on

machine M_2. We denote by $p_{k,j}$ the processing time of job j on machine k ($j = 1, ..., n$, $k = 1, ..., 2$). Also, we denote by $C_{k,j}$ the completion time of job j on machine M_k. Preemption on either machine is not allowed. The objective is the minimization of the sum of completion times on the second machine. It is well known for this problem that there exists at least one optimal solution which is a permutation schedule, that is the jobs share the same sequence on all machines. This problem is known to be NP-hard in the strong sense. We denote by $F2 \mid\mid \sum C_j+$ the re-optimization of version of $F2 \mid\mid \sum C_j$ where a single job x with processing times p_{1x} and p_{2x} is added to an instance already solved to optimality. We analyze the approximation ratios of simple re-optimization strategies that keep unchanged the baseline schedule, namely that merely insert the new job into the initial optimum. We denote by $\mathtt{A}_{i,j}$ ($1 \leq i \leq j \leq n+1$) the reoptimization strategy that consists of computing $j-i+1$ candidate solutions, by inserting the new job in the initial optimum in the k^{th} position for all possible k's between i and j, while leaving the rest of the scheduling unchanged. We provide here the findings of [6] on this problem. Without loss of generality, the initial optimum is given by schedule $[1, ..., n]$ and has value $f(OPT) = sum_{j=1}^{n} C_{2j}$. On the other hand, the unknown optimal sequence on the modified instance is denoted by OPT' and its value is denoted by $f(OPT')$.

The following propositions (proposed in [6]) hold.

Proposition 1. *$F2 \mid\mid \sum C_j+$ is approximable within ratio $\frac{3}{2}$ by algorithm $\mathtt{A}_{n,n+1}$ and this bound is tight.*

Proposition 2. *The algorithm $\mathtt{A}_{1,n+1}$ cannot ensure approximation ratio better than $\frac{7}{5}$ for the $F2 \mid\mid \sum C_j+$ problem.*

In order to show that the tightness of the ratio $\frac{3}{2}$ for algorithm $\mathtt{A}_{n,n+1}$ in Proposition 1 and to prove Proposition 2, we need to derive the related instances. This issue can be tackled by means of LP modeling. The rationale of the approach is as follows. Let denote by $S1_h$ for $h = 1, \ldots, k!$ all permutations of k jobs and let assume without loss of generality that $S1_1 = [1, \ldots, k]$ is optimal, that is $f(S1_1) = OPT$. On the other hand let denote by $S2_h$ for $h = 1, \ldots, (k+1)!$ all permutations of $k+1$ jobs (the first k jobs of the original sequence plus job x).

The proposed approach uses as as constraints the requirement that the initial optimal schedule is $S1_1$ (namely $f(S1_h) \geq f(S1_1)$ for $h = 2, \ldots, k$) and the requirement that all schedules reachable by algorithms $\mathtt{A}_{n,n+1}$ (for the instance of Proposition 1) or $\mathtt{A}_{1,n+1}$ (for the instance of Proposition 2) cannot have a solution value inferior to some given predefined value τ. Finally the goal is to find a a sequence $S2_h$ for $h = 1, \ldots, (k+1)!$ such that $f(S2_h)$ is minimum. The variables are then the jobs processing times (corresponding to $2(k+1)$ variables) and the jobs completion times induced by each sequence (corresponding to $2k \cdot k!$ variables for sequences $S1_h$ and to $2(k+1) \cdot (k+1)!$ variables for sequences $S2_h$). For any given sequence $S2_\gamma$, this is accomplished by solving a related ILP model.

Let $p(i, j)$ be the processing time of job j on machine i. Let $p(h, i, [j])$ be the processing time of the $j-th$ job on machine i according to sequence h. Let $C(h, i, [j])$ be the completion time of the $j-th$ job on machine i according

to sequence h. Let $f(S2_\gamma)$ denote the sum of completion times on the second machine of sequence $S2_\gamma$. All these are integer variables. The ILP model is as follows.

$$\min f(S2_\gamma) = \sum_{j=1}^{k+1} C(S2_\gamma, 2, [j]) \tag{12}$$

$$\sum_{j=1}^{k} C(S1_h, 2, [j]) \geq \sum_{j=1}^{k} C(S1_1, 2, [j]) \; \forall h = 2 \dots k! \tag{13}$$

$$\sum_{j=1}^{k} C(S2_h, 2, [j]) \geq \tau \; \forall h \in Q \tag{14}$$

$$C(S1_h, 1, [1]) = p(S1_h, 1, [1]) \; \forall h = 1 \dots k! \tag{15}$$

$$C(S1_h, 1, [j]) = C(S1_h, 1, [j-1]) + p(S1_h, 1, [j]) \; \forall h = 1 \dots k! \; \forall j = 2 \dots k \tag{16}$$

$$C(S1_h, 2, [1]) = C(S1_h, 1, [1]) + p(S1_h, 2, [1]) \; \forall h = 1 \dots k! \tag{17}$$

$$C(S1_h, 2, [j]) = \max\{C(S1_h, 1, [j]), C(S1_h, 2, [j-1])\} + p(S1_h, 1, [j]) \tag{18}$$

$$\forall h = 1 \dots k! \quad \forall j = 2 \dots k \tag{19}$$

$$C(S2_h, 1, [1]) = p(S2_h, 1, [1]) \; \forall h = 1 \dots k+1! \tag{20}$$

$$C(S2_h, 1, [j]) = C(S2_h, 1, [j-1]) + p(S2_h, 1, [j]) \; \forall h = 1 \dots k+1! \; \forall j = 2 \dots k \tag{21}$$

$$C(S2_h, 2, [1]) = C(S2_h, 1, [1]) + p(S2_h, 2, [1]) \; \forall h = 1 \dots k+1! \tag{22}$$

$$C(S2_h, 2, [j]) = \max\{C(S2_h, 1, [j]), C(S2_h, 2, [j-1])\} + p(S1_h, 1, [j]) \tag{23}$$

$$\forall h = 1 \dots k+1! \quad \forall j = 2 \dots k \tag{24}$$

$$\text{matching constraints binding } p(h, i, [j])\text{'s with } p(i, j)\text{'s} \quad \forall h, \forall i, \forall j \tag{25}$$

Here constraints (13) indicate that $S1_1$ is the optimal sequence for the original problem. Also, in (14), Q is the subset of schedules whose cost function value must be $\geq \tau$ as indicated in Propositions 1 and 2. Then, (15–24) are the standard sequencing constraints between adjacent jobs on any given machine or between adjacent operations of the same job. Notice that here for conciseness we kept in constraints (18,19) and (23,24) a non linear max notation that can be easily transformed into linear constraints by means of $big - M$ coefficients and the introduction of 0/1 variables. Finally, constraints (25) link together the $p(h, i, [j])$ variables with the $p(i, j)$ ones (for instance, if in $S1_1$ job 2 is in first position, then $p(S1_1, i, [1]) = p(i, 2) \; \forall i = 1, 2$).

Then, by iterating the solution of the LP model on all $\gamma = 1, \dots, (k+1)!$ and taking the minimum value, we get the best possible instance (with $k+1$ jobs where job $k+1$ corresponds to job x). Notice, however, that the size k must be strongly limited.

By means of the above approach, it was possible to derive for Proposition 1 the following original instance with $k = 2$. Here we have job 1 that has $p_{1,1} = 0$ and $p_{2,1} = 1$ and job 2 that has $p_{1,2} = 1$ and $p_{2,2} = 0$. For this instance, the optimal scheduling is $[1, 2]$ and has value 2. Then, the new job x added has $p_{1,x} = 0$ and $p_{2,x} = 0$. In this case, it is easy to see that the optimal sequence for the new instance with job x is $[x-1-2]$ with value 2, while algorithm $\mathtt{A}_{n,n+1}$ reaches in the best case value 3.

On the other hand, it was possible to derive for Proposition 2 the following original instance still with $k = 2$. Here we have job 1 that has $p_{1,1} = 0$ and $p_{2,1} = 2$ and job 2 that has $p_{1,2} = 1$ and $p_{2,2} = \delta$. Again, for this instance, the optimal scheduling is $[1, 2]$ and has value $4 + \delta$. Then, the new job x added has $p_{1,x} = 0$ and $p_{2,x} = 1$. In this case, it is easy to see that the optimal sequence for the new instance with job x is $[x - 2 - 1]$ with value $5 + 2\delta$, while algorithm $\mathtt{A}_{1,n+1}$ reaches in the best case value $7 + \delta$.

4 Mathematical programming modeling for the measure-and-conquer paradigm in exact branch-and-reduce algorithms

The design of exact methods for NP-hard combinatorial optimization problems has always been a challenging issue. Among the existing exact methods, search tree algorithms have been widely applied. Here, we study their application in the context of worst-case analysis of exact algorithms where classical search tree algorithms are more commonly defined as branch-and-reduce algorithms as typically a branch in the search tree induces the generation of two or more subproblems each with a reduced number of variables with respect to the original problem. Consider now a combinatorial optimization problem that can be represented by means of n binary variables. Let $T(\cdot)$ be a super-polynomial and $p(\cdot)$ be a polynomial, both on integers. In what follows, using notations in [22], for an integer n, we express running-time bounds of the form $p(n) \cdot T(n)$ as $O^*(T(n))$, the asterisk meaning that we ignore polynomial factors. We denote by $T(n)$ the worst case time required to exactly solve the considered combinatorial optimization problem with n binary variables. We recall (see, for instance, [9]) that, if it is possible to bound above $T(n)$ by a recurrence expression of the type $T(n) \leq \sum T(n - r_i) + O(p(n))$, we have $\sum T(n - r_i) + O(p(n)) = O^*(\alpha(r_1, r_2, \ldots)^n)$ where $\alpha(r_1, r_2, \ldots)$ is the largest zero of the function $f(x) = 1 - \sum x^{-r_i}$. As an example, consider an ILP model with only binary variables and suppose that a branching scheme can be constructed such that at each branch at least either 1 or 3 variables can be fixed. Then, we have $T(n) \leq \sum T(n-1) + T(n-3) + O(p(n))$ and we get $T(n) = O^*(\alpha^n)$ with $\alpha =$ largest real root of $1 - \frac{1}{\alpha} - \frac{1}{\alpha^3} = 0$, that is $\alpha^3 = \alpha^2 + 1$, namely $\alpha \approx 1.4656$ and hence $T(n) = O^*(1.4656^n)$. This value can actually be expressed by the following mathematical programming model

$$\min \alpha \tag{26}$$

$$\alpha^3 - \alpha^2 \geq 1 \tag{27}$$

$$\alpha \geq 0 \tag{28}$$

In the context of branch-and-reduce algorithms, the measure-and-conquer paradigm has been proposed in [10] that is based upon a more elaborated design of a non-straightforward measure of the subproblems sizes. Such measure is used to lower bound the progress made by the algorithm at each branching step and may exploit behaviors of the algorithm that a standard measure might not be able to get leading typically to an improved worst-case time analysis. Here, we analyze a standard branch-and-reduce algorithm for the maximum independent set (MIS) problem and how it can be strongly improved by means of the measure-and-conquer paradigm that embeds to this extent the solution of a non linear mathematical programming model. In the MIS problem, given a graph $G(V, E)$, we search for a maximum cardinality subset $I \subset V$ of vertices that does not induce any edges. It is well known that the MIS problem is polynomially solvable if all vertices have degree ≤ 2 ($\forall i \in V \;\; d_i \leq 2$). Alternatively, G contains a vertex j of degree $d_j \geq 3$. Consider a standard branch-and-reduce exact algorithm where a classical binary branching scheme on some vertex j where j is either selected or discarded. The only peculiarity is that we repeat

branching on the largest degree vertex j, that is $d_j = d_{\max}$ until $d_j \leq 2$, $\forall j$ (once $d_j \leq 2$, $\forall j$, the remaining problem is solved to optimality in polynomial time). Then at each branch we have

1. $j \notin I$;
2. $j \in I \longrightarrow I$ cannot contain any neighbor of v.

In the first case, one vertex is fixed, while in the second case $d_j + 1$ vertices are fixed (j plus all neighbors of j). As $d_j \geq 3$, the worst case occurs for $d_j = 3$. Hence, the induced recursion is $T(n) \leq T(n-1) + T(n-4) + O(p(n))$. Correspondingly, by solving the the related mathematical programming model

$$\min \alpha \tag{29}$$

$$\alpha^4 - \alpha^3 \geq 1 \tag{30}$$

$$\alpha \geq 0 \tag{31}$$

we get $T(n) = O^*(1,3803^n)$.

In the above branch-and-reduce algorithm, for each level of the search tree we define as *fixed* those vertices that have already been selected or discarded, while we define as *free* the other vertices.

Now. consider using the measure-and-conquer paradigm. We do not count in the measure the fixed vertices, and we count with weight $w_h = w_{[d_h]}$ the free vertices h where the vertex j to be selected for branching is the one with largest degree that is with largest weight. We take also into account the known fact that, whenever in the search tree we encounter a subproblem where a vertex h has degree 1, then h can be included without loss of optimality in I and its neighbor can be discarded. Then, we impose

$$0 = w_{[1]} \leq w_{[2]} \leq w_{[3]} \leq \ldots \leq w_{[6]} \ = w_{[n]} = 1 \tag{32}$$

that is the weights of the free vertices are strictly increasing in their degrees and that all free vertices with degree ≥ 6 have weight $= 1$ (while $w_{[1]} = 0$ as vertices with just one neighbor can be immediately fixed. We so get recurrences on the time $T(p)$ required to solve instances of size p, where the size of an instance is the sum of the weights of the vertices. Since initially $p = n$, the overall running time is expressed as a function of n. This is valid since when $p = 0$, all vertices have been fixed. Now consider the branching. If we discard j, vertex j is fixed and d_j vertices (the neighbors of j) decrease their degree and correspondingly their weight by one unit. On the other hand, if we select j, d_j other vertices (the neighbors of j) are discarded, that is $d_j + 1$ vertices are fixed. Hence, the recurrence becomes:

$$T(p) \leq T\left(p - w_{[d_{\max}]} - \sum_{h \in N(j)} \left(w_{[d_h]} - w_{[d_h - 1]}\right)\right) + T\left(p - w_{[d_{\max}]} - \sum_{h \in N(j)} w_{[d_h]}\right) \tag{33}$$

To compute the worst-case complexity under the measure-and-conquer paradigm, we have to compute the best value of the weights $w_{[2]}, \ldots, w_{[n]}$ such that recurrence (33) is respected for any possible value of $d_{\max}$ with $3 \leq d_{\max} \leq n$ as the value of $d_{\max}$ may change in the various branches of the search tree.

To avoid a combinatorial explosion of the different subcases of (33) with respect to the possible degrees of the neighbors of the vertex j selected for branching, we impose that

$$w_{[j]} - w_{[j-1]} \leq w_{[j-1]} - w_{[j-2]},\ j = 3, \ldots, n. \tag{34}$$

where we recall from (32) that $w_{[1]} = 0$ and $w_{[6]} = \ldots = w_{[n]} = 1$. Also, we slightly modify the recurrence. Indeed, from (32) we have that

$$\sum_{h \in N(j)} w_{[d_h]} \geq \sum_{h \in N(j)} w_{[2]} \tag{35}$$

always holds as the smallest weight greater than zero is w_2. Finally, from (32), (34) and (35), we have that the following recurrence always holds

$$T(p) \leq T\left(p - w_{[d_{\max}]} - d_{\max}\left(w_{[d_{\max}]} - w_{[d_{\max}-1]}\right)\right) + T\left(p - w_{[d_{\max}]} - \sum_{h \in N(j)} w_{[2]}\right) \tag{36}$$

where there is no need to consider the recurrence for $d_{\max} \geq 8$ as $w_{[k]} - w_{[k-1]} = 0$ for $k \geq 7$. All together, we need to guarantee that recurrences (36) for $7 \geq d_{\max} \geq 3$, and constraints (32) and (34) are satisfied simultaneously.

This corresponds to a non linear programming model of the form:

$$\min \alpha \tag{37}$$

$$\alpha^{w_{[j]}+j*w_{[2]}} - \alpha^{(j*w_{[2]}-j*(w_{[j]}-w_{[j-1]}))} \geq 1, \qquad j = 3, \ldots, 7 \tag{38}$$

$$w_{[j]} \leq w_{[j+1]}, \qquad j = 2, \ldots, 5 \tag{39}$$

$$w_{[6]} = w_{[7]} = 1, \tag{40}$$

$$w_{[j]} - w_{[j-1]} \leq w_{[j-1]} - w_{[j-2]}, \quad j = 3, \ldots, 6 \tag{41}$$

$$\alpha \geq 0. \tag{42}$$

where constraints (38) correspond to recurrences (36), constraints (39), (40) correspond to constraints (32) and constraints (41) correspond to constraints (34) respectively. The corresponding optimal solution is $\alpha = 1.3496$. Thus, we have $T(n) = O^*(1.3496^n)$. Notice that if the recurrence (33) is kept for $d_{\max} = 3, 4$ with all possible related (4 for $d_{\max} = 3$ and 15 for $d_{\max} = 4$) subcases, while recurrence (36) is used for $d_{\max} \geq 5$, then we obtain $T(n) = O^*(1.3186^n)$. Notice also that the currently best available exact algorithm (that uses the measure-and-conquer paradigm) for the maximum independent set problem (see [3] reaches $T(n) = O^*(1.2114^n)$.

5 Constructing logic puzzles by ILP modeling

Following the description of Wikipedia the free encyclopedia [19], Sudoku can be described as a logic-based, combinatorial, number-placement puzzle. The aim of the classic version of the puzzle is to enter a numerical digit from 1 through n (typically $n = 9$) in each cell of a nXn grid made up of $\frac{n}{3}X\frac{n}{3}$ subgrids (often

called blocks - there are then 9 blocks), starting with various digits given in some cells (the givens); each row, column, and region must contain only one instance of each numeral. Further only proper puzzles are considered, that is puzzles having a unique solution. Below are reported a classic $9X9$ Sudoku proper puzzle and the corresponding solution.

				6		1	7	3
	2		5		3			
7						2		
4			6					9
2								8
5					4			2
		6						5
			8		6		4	
8	4	7		9				

9	8	5	4	6	2	1	7	3
6	2	1	5	7	3	8	9	4
7	3	4	1	8	9	2	5	6
4	7	3	6	2	8	5	1	9
2	6	9	7	5	1	4	3	8
5	1	8	9	3	4	7	6	2
1	9	6	2	4	7	3	8	5
3	5	2	8	1	6	9	4	7
8	4	7	3	9	5	6	2	1

The Sudoku puzzle was invented in Indianapolis in 1979 but reached widespread international popularity just in 2005 after being launched at the end of 2004 by one of the leading british newspapers, "The Times". A nice scientific survey of the Sudoku phenomenon was presented in [5]. Several variants have been proposed of the Sudoku puzzle. We cite among others SudokuX, where as additional constraint it is required that also the main diagonals must contain only one instance of each numeral. Alphabetical variations have also emerged (the so-called wordoku): there is no functional difference in the puzzle unless the letters spell something. Some variants include one or two words reading somewhere in the grid once solved, where this/these word/s are described as in crosswords: determining the word in advance can be viewed as a solving aid.

5.1 Solving proper Sudoku puzzles by means of ILP modeling

Every proper Sudoku puzzle can be tackled by means of various AI & OR techniques. For instance, on one hand the puzzle can be solved by means of constraint programming techniques such as the *all-different* operator, on the other hand it can be easily formulated as an ILP model. Here we focus on this latter approach. Given an initial grid where some elements (i, j) have already been filled by digits, let $BL - h$ $(h = 1, ..., 9)$ be the h-th block of the grid. An ILP formulation $(P1)$ of the SUDOKU puzzle can be expressed as follows:

Variables (all variables are binary):

$x_{i,j,k} = 1$ if element (i, j) has value k $(1 \leq k \leq n,\ k$ integer) in the puzzle solution, else $x_{i,j,k} = 0$.

Objective function:

there is no objective function as the purpose is just to search for a feasible solution.

Constraints:

$$\sum_{k=1}^{n} x_{i,j,k} = 1 \quad \forall i, j \tag{43}$$

(each element of the grid contains one of the digits 1-n)

$$\sum_{j=1}^{n} x_{i,j,k} = 1 \quad \forall i, k \tag{44}$$

(each row of the grid contains the digits 1-n exactly once)

$$\sum_{i=1}^{n} x_{i,j,k} = 1 \quad \forall j, k \tag{45}$$

(each column of the grid contains the digits 1-n exactly once)

$$\sum_{i,j:(i,j)\in BL-h} x_{i,j,k} = 1 \quad \forall h = 1, ..., 9 \;\; \forall k = 1, ..., n \tag{46}$$

(each block of the grid contains the digits $1 - n$ exactly once)

$$x_{i,j,k} = 1 \qquad \forall \text{ element of the entry grid } (i, j) \text{ with value } k. \tag{47}$$

5.2 Building proper Sudoku puzzles by means of ILP modeling

To build a proper Sudoku puzzle, we need to derive an initial grid such that there exists a feasible Sudoku solution compatible to that grid and this solution is unique. Below, is indicated how to check whether a solution (given an initial grid) is or not unique. Let denote by $SOL(i, j)$ the value of element (i, j) of the grid in the feasible solution. Consider solving the following ILP model $(P2)$:

Objective function:

$\min Z = \sum_{i,j,k:SOL(i,j)=k} x_{i,j,k}$
(we minimize the sum of all variables $x_{i,j,k}$ corresponding to elements (i, j) having value k in the feasible solution).

Constraints: the same constraints (43)–(47) considered in model P_1.

The solution of model $P2$ clearly provides a feasible solution also to model P_1 and therefore to the SUDOKU puzzle. Further, the objective function of model P_2 minimizes the sum of the elements in the grid having the same value obtained in the solution of model P_1. Hence, as each grid is composed by n^2 elements, if the objective function value of model P_2 is equal to n^2 ($Z = 81$ for $n = 9$), then the considered SUDOKU puzzle has unique solution, else the solution to the SUDOKU puzzle is not unique.

A straightforward approach for building a proper Sudoku puzzle is provided by the following procedure (notice that the sequence indicated below for emptying the elements is not compulsory, but it is sufficient to handle all elements under any sequence).

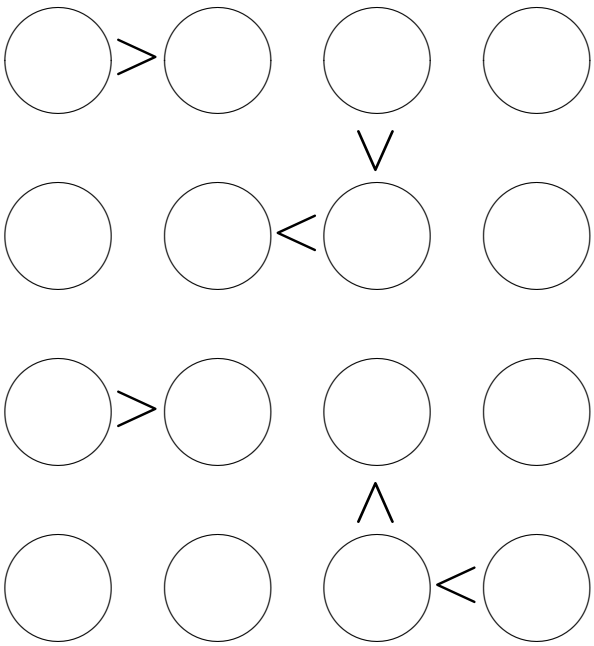
```
Procedure "Build_initial_grid":
INPUT: final solution.
OUTPUT: initial grid.
{
   for i=1 to n
      for j=1 to n
         fill element (i,j) with Sol(i,j);

   for i=1 to n
      for j=1 to n
         empty element (i,j);
         Solve model P2;
         if the objective function value of model P2 is < n x n
         (solution not unique),
         then
            fill element (i,j) with Sol(i,j);
      end for
   end for
}
```

This approach guarantees that the corresponding Sudoku puzzle is also irreducible, namely no element(i,j) can be emptied, or else the solution becomes not unique.

5.3 Futoshiki

In a similar way, we can deal with several other logic puzzle games, such as, for instance, Futoshiki [11]. The puzzle is played on a square grid nXn (typical sizes are $5X5$, $6X6$, $7X7$). The objective is to place the numbers 1 to n (or whatever the dimensions are) such that each row, and column contains each of the digits 1 to n. Some digits may be given at the start. In addition, inequality constraints are also initially specified between some of the squares, such that one must be higher or lower than its neighbour. These constraints must be honoured as the grid is filled out. As for Sudoku, only proper puzzles are considered, that is puzzles having a unique solution. Below are reported a classic $4X4$ Futoshiki proper puzzle and the corresponding solution.



3	2	4	1
2	1	3	4
4	3	1	2
1	4	2	3

Here the same approach indicated for Sudoku with models P_1 and P_2 and Procedure "'Build_initial_grid"' holds. The only difference is that there are no more subgrids but there are inequalities between several adjacent elements of the grid. Correspondingly, the same $x_{i,j,k}$ variables are used. Then in model P_1, constraints (43), (44), (45), (47) are kept while constraint (46) is substituted by constraint

$$\sum_{k=1}^{N} k * x_{i,j,k} \geq \sum_{k=1}^{N} k * x_{l,m,k} + 1 \ \forall i,j,l,m \ \text{ such that } \ (i,j) > (l,m) \qquad (48)$$

this latter constraint indicating that all inequalities present in the puzzle must be respected. Similar consideration holds for model P_2. Finally, Procedure "'Build_initial_grid"' modifies as follows.

```
INPUT: final solution.
OUTPUT: initial grid
{
   for i=1 to n, j=1 to n-1
     if Sol(i,j) > Sol(i,j+1), then add (i,j) >= (i,j+1) + 1;
     else  ADD (i,j) + 1 <= (i,j+1);

   for i=1 to n, j=1 to n-1
      if Sol(i,j) > Sol(i+1,j), then add (i,j) >= (i+1,j) + 1;
      else add (i,j) + 1 <= (i+1,j);

   for all inequalities (randomly sorted) do
      - delete the inequality;
```

```
      - solve model P2;
      - if the objective function value of model P2 is < n x n
        (solution not unique),
then recover the inequality.
   end-do
}
```

6 Conclusions

We have considered in this work the application of ILP and MP modeling in various fields related to combinatorial optimization with emphasis on scheduling problems. We have seen that the one-machine total completion time scheduling problem subject to release times can be satisfactorily solved by means of on an ILP model when valid cuts based on the problem structure are added. Also, we have discussed the application of ILP modeling for tightening bounds in approximation results. Notice that such approach presented for a scheduling problem under a re-optimization framework can be easily generalized to sequencing problems where the sequences reached by the proposed algorithm can be easily detected. Then, an example of MP modeling has been provided within the so-called measure-and-conquer paradigm for exact exponential branch-and-reduce paradigm. Finally, we have shown that several well known logic puzzles such as Sudoku and Futoshiki, but also many others, can be easily constructed by means of a simple approach requiring the iterated solutions of ILP models.

We believe that ILP (and MP) modeling remains a fantastic tool for combinatorial optimizers. Nonetheless several steps are still to come, for instance the generation of an efficient ILP model for the well known minimum makespan jobshop problem ($J||C_{\max}$) remains a formidable challenge for the OR community!

References

[1] Baker K.R., *"Introduction to Sequencing and Scheduling "*, John Wiley, New York, 1974.

[2] Baptiste Ph., Della Croce F., Grosso A., Tkindt V. Sequencing a single machine with due dates and deadlines: an ILP-based approach to solve very large instances. Journal of Scheduling, 13, 3947, 2010.

[3] Bourgeois N., Escoffier B., Paschos V.Th., van Rooij J.M.M. Fast algorithms for MAX INDEPENDENT SET. Algorithmica, 62, 382-415, 2012.

[4] Chu C., A branch–and–bound algorithm to minimize total flow time with unequal release dates. Naval Research Logistics, 39, 859-875, 1992.

[5] Delahaye J.P., The Science behind Sudoku. Scientific American, 71-77, 2006.

[6] Boria N., Della Croce F., Re-optimization in Machine scheduling. EURO-IFORS 2013 International Conference, Rome, Italy, July 2013.

[7] Della Croce F., Salassa F., T'kindt V., A Hybrid Heuristic Approach for Single Machine Scheduling with Release Times. *submitted*.

[8] Della Croce F., T'kindt V., Improving the preemptive bound for the one-machine dynamic total completion time scheduling problem. Operations Research Letters, 31, 142-148, 2003.

[9] Eppstein D., Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In Proc. Symposium on Discrete Algorithms, SODA01, 329-337 (2001).

[10] Fomin F.V., Grandoni F., Kratsch D. A measure and conquer approach for the analysis of exact algorithms. Journal of the ACM, 56, 1-32, 2009.

[11] Futoshiki, from Wikipedia: *http://en.wikipedia.org/wiki/Futoshiki*.

[12] Jouglet A., Savourey D., Carlier J., Baptiste Ph. Dominance-based heuristics for one- machine total costs scheduling problems. European Journal of Operational Research, 184, 879-899, 2008.

[13] Lasserre J.B., Queyranne M., Generic scheduling polyhedral and a new mixed integer formulation for single machine scheduling. in Proceedings of the IPCO Conference, 136-149, 1992.

[14] Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Shmoys D.B., Sequencing and scheduling: algorithms and complexity. In: Graves SC, Rinnooy Kan AHG and Zipkin PH (eds)., Logistics of Production and Inventory. North Holland, Amsterdam, Netherlands, 445-522, 1993.

[15] MHallah, R., Bulfin, R. L. Minimizing the weighted number of tardy jobs on a single machine. European Journal of Operational Research, 145, 45-56, 2003.

[16] Plastria F., Formulating logical implications in combinatorial optimisation. European Journal of Operational Research, 140, 338-353, 2002.

[17] Smith S.S., Reactive scheduling systems. In D.E. Brown and W.T. Scherer, editors. Intelligent Scheduling Systems, Kluwer Academic Publishers, 155-192, 1994.

[18] T'Kindt V., Della Croce F., Esswein C., Revisiting branch and bound search strategies for machine scheduling problems. Journal of Scheduling, 7, 429-440, 2004.

[19] Sudoku, from Wikipedia: *http://en.wikipedia.org/wiki/Sudoku*.

[20] Tanaka S., Fujikuma, S. A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time, Journal of Scheduling, 15, 347-361, 2012.

[21] Williams H.P., Model Building in Mathematical Programming. Wiley, 1990.

[22] Woeginger, G. J., Exact algorithms for NP-hard problems: a survey. In M. Juenger, G. Reinelt, and G. Rinaldi, (eds.) Combinatorial Optimization - Eureka! You shrink!, volume 2570 of Lecture Notes in Computer Science, 185-207, Springer-Verlag, 2003.

Author Index

Abdullah S. 795
Ackermann H. 617
Adasme P. 55
Aggoune-Mtalaa W. 753
Aghezzaf E.-H 230, 676
Agnetis A. 133
Aharon G. 800
Alefragis P. 392
Allaoui H. 240
Alonso-Pecina F. 812
Aloulou M.A. 133
Amrouche K. 610
Artiba H. 240
Artigues C. 356, 815
Asta S. 836
Audenaert P. 711
Ayob M. 549
Barbosa A. 151
Barbosa J.J.G. 472
Bargiela A. 528
Barth T.C. 599
Beliën J. 580, 633, 737
Bellenguez-Morineau O. 590
Bellio R. 314
Bello R. 410
Benavides A.J. 826
Benmansour R. 240
Berghman L. 603
Berlinska J. 80
Bessler S. 486
Bierwirth C. 613
bin Asmuni H. 543
Bitar A. 718
Blazewicz J. 667
Böckmann F. 599
Bocquillon R. 587
Borba L.M. 826
Borodin D. 402
Boudhar M. 610
Bouly H. 831
Bouquard J-L. 680
Bouvry P. 472, 667
Brauner N. 790
Briand C. 125
Bucar D. 486
Burke E. 528

Bykov Y. 691
Carniel G.C. 826
Carvalho M. 840
Cerdeira J.O. 620
Ceschia S. 314, 646
Chen M. 573
Cheng C. 593
Chrétienne P. 39, 206
Christiaens J. 786
Clark M.D. 767
Colen P. 509
Coolen K. 670
Cortés R. 748
Costa e Silva E. 620
Creemers S. 509
Curran D. 606
Dang D-C. 831
Dauzère-Pérès S. 687, 718, 741
Davari M. 656
De Bruecker P. 633
De Bruyn W. 402
De Causmaecker P. 21
De Feyter T. 424
De Wispelaere A. 786
Deghdak K. 680
Deleplanque S. 185
Della Croce F. 13, 849
Demeester P. 711
Demeulemeester E. 633, 656, 774
Demir O. 372
Di Gaspero L. 314
Dolgui A. 290, 636
Dostert M. 218
Drozdowski M. 80
Earl M. 781
Elhag A. 770
Ermis G. 584
Fernandes P. 151
Finke G. 790
Fonseca G.H.G. 331, 345
Fu L-L. 133
Gafarov E.R. 290, 636
Gagné C. 17
García M.J.G. 694
Garcia-Sabater J.P. 515, 748
Garraffa M. 703
Geiger M.J. 807
Geirsson E. 318
Gogos C. 392
Gonzales C. 268
Goossens D. 596

Goulas G. 392
Gravel M. 17
Groenbaek J. 486
Guedes A.F.M., ... 707
Guerry M-A. 424
Hamilton-Bryce R. 499
Han J. 733
Hanafi S. 240
Hanen C. 721
Hanzalek Z. 721, 744
Hazir O. 757
He R. 593
Hebrard E. 815
Hernández J.G.R. 694
Hernández J.J.G. 694
Hildebrandt T. 437
Houbraken M. 711
Housos E. 392
Huacuja H.J.F. 472
Ingels J. 684
Jeauneau V. 206
Jouglet A. 587
Karapetyan D. 836
Kedad-Sidhoum S. 268, 757
Kellerer H. 577
Kendall G. 549
Kheiri A. 836
Klos T. 555
Kochaniková B. 762
Komarudin 424
Kristiansen S. 628
Küfer K-H. 617
Kuhpfahl J. 613
Laborie P. 379
Lamas P. 774
Lambrecht M. 509
Lancia G. 649
Lapègue T. 590
Lazarev A.A. 290, 636
Leoff J. 617
Leus R. 580, 624, 656, 670
Limère V. 230
Lisser A. 55
Little J. 606
Lopes I.C. 620
López-Ibáñez M. 820
Lykourentzou I. 753
Maenhout B. 673, 684
Marmion M-E. 820
Martínez Y. 410
Mascia F. 820

Mason A. 628
Mateo M. 659
McCollum B. 499, 528, 543, 777, 795
McMullan P. 499
Mercy A. 737
Messelis T. 21
Mihaylov M. 725
Moench L. 687
Molina P. 748
Moukrim A. 831
Moura A. 111
Muehlenthaler M. 300
Muklason A. 777
Müller T. 248
Musial J. 667
Musliu N. 486
Nakamura S. 573
Nakayama K. 573
Naudet Y. 753
Nazri M.Z.A 641
Neufeld J.S. 741
Ngueveu S.U. 125
Nishi T. 65
Nobibon F.T. 580, 624, 656, 670
Nowé A. 402, 410
Oguz C. 584
Oliveira J. 111
Özcan E. 528, 770, 777, 836
Özdamar L. 372
Parkes A.J. 767, 777, 836
Pecero J.E. 472, 667, 812
Pereira C.S. 151
Petrovic S. 691
Pickavet M. 711
Pineda A.A.S. 472
Pira C. 356
Politz A. 218
Potts C. 14
Prot D. 590
Quilliot A. 39, 91, 185
Rahman S.A. 528
Ritt M. 826
Rodrigues L.U. 458
Rogerie J. 379
Romero D. 812
Rudová H. 762
Runarsson T.P. 318
Sabar N.R. 549, 641, 795
Sadykov R. 733
Sahin G. 729
Salassa F. 703

Santos H.G. 331, 345, 458, 840
Saridarq F.D. 729
Schaerf A. 314, 646
Schäfer A. 599
Schmitz H. 218
Schnell A. 802
Scholz-Reiter B. 437
Seddik Y. 268
Serafini P. 649
Shafransky Y. 790
Shen L. 741
Silva C.L.T.F. 458
Sioud A. 17
Sledneu D. 790
Smet P. 664
Soares J. 840
Soto I. 55
Spieksma F.C.R. 596, 603
Stidsen T.R. 628
Strusevich V. 577
Stützle T. 820
Sucha P. 125, 744
Sugiyama T. 65
Tan Y. 437
Tanaka S. 171
Tereso A.P. 707
T'Kindt V. 603, 680
Toffolo T.A.M. 331, 458, 840
Toussaint H. 91
Trystram D. 667
Turky A.M. 795
Urli T. 314
Vaclavik R. 744
Valero-Herrero M. 515
Valouxis C. 392
Van den Bergh J. 580, 633
Van der Krogt R. 606
Van Vreckem B. 402
Vancroonenburg W. 652, 703
Vanden Berghe G. 21, 424, 664, 703, 725, 786
Vanderbeck F. 733
Vanhoucke M. 673
Vansteenwegen P. 659, 676
Verbeeck C. 676
Vergados D.J. 753
Vidal-Carreras P.I. 515
Voros N. 392
Walcarius T. 711
Walker S. 714
Wang C. 55
Wanka R. 300

Wauters T. 652, 703, 725, 786
Wei W. 624
Weng F.C. 543
Werner F. 290
Wilson M. 555
Wilson N. 606
Witteveen C. 15, 555
Xu H. 831
Yang Z. 593
Yao F. 593
Yugma C. 718
Zeltzer L. 230
Zinder Y. 714
Zubaran T. 826