

MISTA 2011

Proceedings of the 5th Multidisciplinary International Conference on Scheduling: Theory and Applications

9th - 11th August 2011 (Tempe Mission Palms, Arizona, USA)

Edited by:

- John Fowler, Arizona State University, USA
- Graham Kendall, University of Nottingham, UK
- Barry McCollum, Queens University Belfast, UK

MISTA 2011 Conference Program Committee

Abdullah, Salwani
Allahverdi, Ali
Alvarez-Valdes, Ramon
Artigues, Christian
Ayob, Masri
Bai, Ruibin
Baker, Ken
Barták, Roman
Bini, Enrico
Blazewicz, Jacek
Brucker, Peter
Burke, Edmund
Cai, Xiaoqiang
Carlier, Jacques
Chen, Zhi-Long
Cheng, Edwin
Coffman, Ed
Cucu-Grosjean, Liliana
Dauzère-Pérès, Stéphane
De Causmaecker, Patrick
Dell'Amico, Mauro
Dowsland, Kath
Dror, Moshe
Drozdowski, Maciej
Finke, Gerd
Fowler, John (Conference Chair)
Froncek, Dalibor
Gendreau, Michel
Glover, Fred
Grabot, Bernard
Hao, Jin-Kao
Henz, Martin
Herrmann, Jeffrey W.
Hoogeveen, Han
Hyde, Matthew
Janiak, Adam
Jouglet, Antoine
Kempf, Karl
Kendall, Graham (Conference Chair)
Kingston, Jeffrey H.
Knust, Sigrid
Kubiak, Wieslaw
Kurz, Mary
Kutanoglu, Erhan

Kwan, Raymond
Lau, Hoong Chuin
Leung, Joseph
Levner, Eugene
Løkketangen, Arne
Mason, Scott
Mattfeld, Dirk
McCollum, Barry (Conference Chair)
McMullan, Paul
Moench, Lars
Mumford, Christine
Norman, Bryan
Ochoa, Gabriela
Osman, Ibrahim
Ozcan, Ender
Pappis, Costas
Pesch, Erwin
Petrovic, Dobrila
Petrovic, Sanja
Pillay, Nelishia
Portmann, Marie-Claude
Prins, Christian
Qu, Rong
Rayward-Smith, Vic J.
Ribeiro, Celso
Rudová, Hana
Sagir, Mujgan
Schaerf, Andrea
Schulz, Andreas S.
T'Kindt, Vincent
Thompson, Jonathan
Trautmann, Norbert
Trick, Michael
Trystram, Denis
Urrutia, Sebastián
Van De Velde, Steef
Vanden Berghe, Greet
Wallace, Stein W
Watson, Jean-Paul
White, George
Yew, Chong Siang
Yugma, Claude
Zinder, Yakov

MISTA 2011 International Advisory Committee

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

Acknowledgements

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

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

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

We are grateful to the special session chairs for taking the time to collect together an excellent set of papers. The sessions/chairs are as follows:

1. **Transport and Logistics** (chaired by Chris Potts)
2. **Healthcare** (chaired by Jonathan Thompson)
3. **Decision-making Under Uncertainty** (chaired by Stein W. Wallace)
4. **Systems to Build Systems** (chaired by Edmund Burke and Ender Ozcan)

All the sessions were arranged as part of the LANCS initiative (see <http://www.lancs-initiative.ac.uk/>) and we also grateful to them for supporting MISTA 2011.

We appreciate the financial support that we received from Gurobi Optimization, LINDO SYSTEMS INC. and Arizona State University.

We are also grateful for the support we have received from the University of Nottingham, and EventMAP Ltd.

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.

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

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

Table of Contents

Plenary Papers

Fordyce K. and Milne J. <i>The Ongoing Challenge – A Responsive Enterprise-Wide Demand Supply Network for Semiconductor Based Package Goods – Something for Everyone</i>	10
K.D. Glazebrook K.D., Hodge D.J. Kirkbride C. and Minty R.J. <i>Stochastic scheduling: A short history of index policies and some key recent developments</i>	94
Whitley D. <i>Quasi-Elementary Landscapes</i>	113

Papers

Borgulya I. <i>An Evolutionary Algorithm for the large Size One-Commodity Pickup and Delivery Travelling Salesman Problems</i>	127
Brandinu G. and Trautmann N. <i>An MILP-Approach to the Optimization of Event-Bus Schedules: A Scheduling Application in the Tourism Sector</i>	137
Atkin J.A.D, Burke E.K. and Ravizza S. <i>A More Realistic Approach for Airport Ground Movement Optimisation with Stand Holding</i>	146
Bai R., Li J., Atkin J.A.D. and Kendall G. <i>A Game Theoretic Approach for Taxi Scheduling Problem with Street Hailing</i>	161
Shufan E., Ilani H. and Grinshpoun T. <i>A Two-Campus Transport Problem</i>	173
Ramacher R. and Mönch L. <i>Hybridized Genetic Algorithms to Schedule Interfering Job Sets on a Single Machine</i>	185
Lin Y.K., Hsieh F.Y. and Lin G.J. <i>Scheduling Identical Jobs on Uniform Parallel Machines to Minimize Maximum Lateness</i>	195
Herrmann J.W. <i>Generating Better Cyclic Fair Sequences Faster with Aggregation.....</i>	200
Atakan S., Tezel B., Bülbül K. and Noyan N. <i>Minimizing Value-at-Risk in the Single-Machine Total Weighted Tardiness Problem.....</i>	215
Vakhania N. <i>Minimizing the Maximal Lateness on a Single-Machine: A New Polynomially Solvable Case</i>	230
Croce F. D. and Koulamas C. <i>Minimizing the Sum of Quadratic Completion Times on Two Identical Parallel Machines.....</i>	240
Hanzálek Z. and Hanen C. <i>Impact of Core Precedences in a Cyclic RCPSP with Temporal Constraints</i>	245
Seddik Y., Gonzales C. and Kedad-Sidhoum S. <i>Single Machine Scheduling with Delivery Dates and Cumulative Payoffs</i>	261

Bispo C.F. <i>Single Server Scheduling Problem: Optimal Policy for Convex Costs Depends on Arrival Rates</i>	275
Dobson R. and Steinhöfel K. <i>Low Energy Scheduling with Power Heterogeneous Multiprocessor Systems</i>	297
Ásgeirsson E.I., Kyngäs J., Nurmi K. and Stølevik M. <i>A Framework for Implementation-Oriented Staff Scheduling</i>	308
Mareček J., Parkes A. J., Burke E.K., Elliot R., Francis H. and Lokhmotov A. <i>Dynamic Data Structures for Taskgraph Scheduling Policies with Applications in OpenCL Accelerators</i>	322
Burke E.K. and Pais T.C. <i>Using Differential Evolution to Identify Fuzzy Measures for the Exam Timetabling Problem</i>	335
Marla L. and Barnhart C. <i>A Decomposition Approach for Shipment Pickup and Delivery with Time-Windows Under Uncertainty</i>	352
Rizzo L.M. and Urrutia S. <i>A GRASP Heuristic to the Extended Car Sequencing Problem</i> ..	366
Misir M., Verbeeck K., De Causmaecker P. and Vanden Berghe G. <i>A New Hyper-Heuristic Implementation in HyFlex: A Study on Generality</i>	374
Ng T.S.A. and Sy C.L. <i>Robust Target Achievement for Decision-Making with Application to Workforce-Inventory Planning</i>	394
Cheng J., Fowler J.W., Hertzler E.F., Huff J.D. and Kempf K.G. <i>Multi-Mode Resource-Constrained Project Scheduling of Semiconductor Equipment Install/Qual Process</i>	408
Ascó A., Atkin J.A.D. and Burke E.K. <i>The Airport Baggage Sorting Station Allocation Problem</i>	419
Adasme P., Lissner A. and Soto I. <i>SDP Relaxations for Two Stage OFDMA Networks with and without Probabilistic Constraints</i>	445
Uhlig T. and Rose O. <i>A Multi Species Evolutionary Algorithm for Tool Group Scheduling in Semiconductor Manufacturing</i>	459
Borodin D., Van Vreckem B. and De Bruyn W. <i>Case Study: An Effective Genetic Algorithm for the Chemical Batch Production Scheduling</i>	469

Abstracts

Mirchandani P. and Li J-Q. <i>Schedule Recovery Strategies in Public and Freight Transportation Systems under Disruptions</i>	484
Khosravi B., Bennell J.A. and Potts C.N. <i>Train Scheduling in the UK using a Job Shop Scheduling Approach</i>	486
Silva-Lugo R., Kozma R., Werbos L. and Pazienza G.E. <i>Optimization of Scheduling for Large-Scale Logistics Tasks</i>	489
Hieu T.T. and Ming N.K. <i>Two-Level Water-Flow Algorithm for Vehicle Routing Problems</i>	493
Xie L., Ma S. Huang P. and Gui H. <i>Supply Logistics Synchronization: Scale Development and Empirical Study of Chinese Manufacturing</i>	497
Shen C-W. and Quadrioglio L. <i>An Insertion Heuristic for Coordinated Zoning Paratransit Systems</i>	502
Akkan C. <i>Improving Schedule Stability in Single-Machine Rescheduling due to New Operation Insertion</i>	506
Shen L., Mönch L. and Buscher U. <i>An Iterative Scheme for Parallel Machine Scheduling with Sequence Dependent Family Setups</i>	519
Lenté C., Liedloff M., Soukhal A. and T'Kindt V. <i>Sort & Search Exponential-Time Algorithms for Scheduling Problems</i>	523
Fleszar K. and Charalambous C. <i>Hybridizing Local Search with MIP for Unrelated Parallel Machine Scheduling Problems with Makespan Minimization</i>	529
Zajiček T. and Šůcha P. <i>Flow Shop Scheduling on GPU</i>	532
Kuhpfahl J. and Bierwirth C. <i>Computational Comparison of Neighbourhoods for the Job Shop Scheduling Problem with Total Weighted Tardiness Objective</i>	536
Baumann P. and Trautmann N. <i>Minimizing Changeover Times of a Make-and-Pack Production Process: Model and Case Study</i>	539
Dimitrov T., Baumann M. and Schenk M. <i>Scheduling of Large Dynamic Job Shop Problems by Concurrent Non-Blocking Threads</i>	542
Sun Y, Li X. and Fowler J.W. <i>Job Scheduling to Minimize Weighted Completion Time Variance on Identical Parallel Machines</i>	546
Smet P, Vancroonenburg W. and Vanden Berghe G. <i>Automated Scheduling at a Joinery Site</i>	549
Bürgy R. and Gröflin H. <i>Optimal Job Insertion in the No-Wait Job Shop</i>	553
Almeder C. and Mönch L. <i>A GRASP Approach for Scheduling Jobs with Incompatible Families on Parallel Batch Machines</i>	557

Čapek R., Šůcha P. and Hanzálek Z. <i>Minimizing Total Weighted Tardiness and Total Processing Costs for the Scheduling with Alternative Process Plans</i>	561
Lozano A.J. and Medaglia A.L. <i>Scheduling of Parallel Machines with Sequence-Dependent Batches and Product Incompatibilities in an Automotive Glass Facility</i>	564
Khowala K., Fowler J., Keha A. and Balasubramanian H. <i>Single Machine Scheduling with Interfering Job Sets to Minimize Total Weighted Completion Time and Maximum Lateness</i>	568
Pfund M.E. and Fowler J.W. <i>Investigating the Boundary between Scheduling and Dispatching</i>	573
Bäumelt Z., Šůcha P. and Hanzálek Z. <i>A Parallel Approach for a Nurse Rostering Problem on the GPU</i>	576
Haspeslagh A. and De Causmaecker P. <i>Pareto Optimal Negotiation for Nurse Rostering Problems</i>	579
Thompson J. and Dowsland K. <i>An Exact and Heuristic Hybrid for Solving a Medical Student Scheduling Problem</i>	584
Herring W.L. and Herrmann J.W. <i>Scheduling Impatient Jobs with Diminishing Rewards: The Impact of Deteriorating Survival Probabilities on Mass Casualty Incident Triage</i>	587
Riise A. and Mannino C. <i>Operational Surgery Scheduling Problems: Modelling and Solving</i>	592
Trautmann N., Baumann P., Saner N. and Schäfer T. <i>Decomposition Approaches for Multi-Stage Multi-Product Batch-Production Scheduling: A Refined Batching Model</i>	596
Pabst D., Fowler J.W. and Mackulak G.T. <i>Flexible Constraint Optimization of Product Profits: A Responsive Approach to Determine Profit-Optimal Product Start Rates Subject to Mean Cycle Time and Product Volume Constraints in Support of Production Planning for Semiconductor Fabrication Facilities</i>	599
Hui Y.V., Gao J., Leung L.C.K. and Wallace S.W. <i>Two-Stage Shipment Planning with Uncertainties: A Stochastic Programming Approach</i>	602
Wallace S.W. and Zhao Y. <i>Facility Layout with Random Demand: A Reduction to the Classical Deterministic QAP</i>	605
Vespucci M.T., Bertocchi M., Innorta M. and Zigrino S. <i>Deterministic and Stochastic Models for Investment in Different Technologies for Electricity Production in the Long Period</i>	607
Donaldson J., McCollum B. and Scott S. <i>Ring Out the Bells</i>	624
Pillay N. <i>A Hyper-Heuristic Approach to Solving School Timetabling Problems</i>	628
Li S., Hajjam A., Créput J.C. and Koukam A. <i>Two Phases Parallel Hyperheuristic Approach to the Bus Driver Scheduling Problem</i>	633

Romeijnders W. and Sierksma G. <i>Off-shore Helicopter Routing with Asymmetric Demands and Refueling Options</i>	637
Park K. and Morrison J.R. <i>Scheduling Wafer Release in Clustered Photolithography Tools</i>	639
Rainwater C.E., Mason S.J., Fowler J.W. and Mönch L. <i>Algorithmic Parallelization in Multi-Criteria Semiconductor Scheduling</i>	642
Obeid A., Dauzère-Pérès S. and Yugma C. <i>Scheduling Job Families on Parallel Machines with a Bi-Criteria Objective Function under Time Constraints</i>	645
Rudová H. and Müller T. <i>Rapid Development of University Course Timetables</i>	649
Kurz M.E. and Sankaran V. <i>Adapting Particle Swarm Optimization for the Flexible Flow Line</i>	653

Plenary Papers

The Ongoing Challenge – A Responsive Enterprise-Wide Demand Supply Network for Semiconductor Based Package Goods – Something for Everyone

Kenneth Fordyce, John Milne

Abstract: Most organizations, from health care facilities to semiconductor manufacturing, can be viewed as an ongoing sequence of loosely coupled decisions *where current and future assets are matched with current and future demand across the demand-supply network at different levels of granularity ranging from placing a lot on a tool (dispatch) to an aggregate capacity plan across a five-year horizon*. By itself, this creates a substantial challenge to management and the computational intelligence community to put in place applications that enable a firm to respond quickly and intelligently to changes in demand and/or assets. The nature of semiconductor manufacturing adds such features as re-entrant flow, alternative bill of materials, repair actions; variability in capacity, long and short lead times, variability in demand, lead time versus utilization trade-off, et al to the “challenge pile”. This presentation will outline the basic playing field and challenges at the global planning level, the factory planning level, the dispatch level, and co-ordination between these three levels. Additionally, it will briefly describe some recent advances.

1. Introduction

It has been approximately 50 years since the advent of modern computing and 60 years since the “general availability” of linear programming as an effective algorithm for planning (matching assets to demand over time). These two events (and a few others) have lead to avalanche of papers, mathematical models, and applications on planning, scheduling, and dispatch (PSD). Despite this work, substantial challenges and opportunities remain! Why?

All models require the adoption of a set of approximations to reality and for most of the last 50 years the core computational environment (hardware, software, and algorithms) and the business culture have imposed some “game” limiting approximations that ignore scope (complexity and variability) and scale (size) in each level of the demand supply network and co-ordination of levels (described in section 2). On the “math” side we can identify such factors as: lack of computational horsepower; cultural influence of the pre 1970 applied mathematics community; lack of access to an experimental proofing ground (the business environment); and the slow pace of diffusion. On the “business culture side”, Ignizio [42] identifies two primary culprits:

- failure to attend to the third dimension of manufacturing – the protocols (policies, practices, and procedures) employed to manage and run the supply chain and factory (the other two dimensions are: physical features and physical components)
- lack of appreciation of the complexity, variability, and scale in the demand supply network

Kenneth Fordyce
IBM Corp.
fordyce@us.ibm.com

R. John Milne
Clarkson University
jmilne@clarkson.edu

However, over the past 15 years the cost, especially in terms of lost responsiveness, of failing to adequately address scope and scale in PSD processes is slowly being recognized in the business community and organizations who have been early leaders have reaped the benefit. Net is:

- “smarter” decisions improve organizational performance
- this requires handling scope (complexity and variability) and scale (size) [52]
- which requires improved computational methods and business process

The paper will progress as follows: overview of the demand supply network; overview of the production of semiconductor based packages goods (in computers, personal devices, security cameras, air bags, washers and dryers, engines, video games, etc); central planning basics and challenges; factory planning basics and challenges; dispatch basics and challenges; co-ordination between the three levels.

The material in this paper is based on our experience and the experience of colleagues. There are many excellent papers in the literature identifying the major challenges- a few are [24, 48, 52, 82, 83]. It is outside the scope of this paper to review them.

2. Enterprise Demand Supply Network – Levels and Components of the Playing Field

Organizations, from health care facilities to manufacturing giants to small restaurants, can be viewed as an ongoing sequence of loosely coupled activities where current and future assets are matched with current and future demand across the demand-supply network.

These planning, scheduling, and dispatch decisions across a firm’s demand-supply network are best viewed as a series of information flows and decision points organized in a decision hierarchy or tiers and further classified by the type of supply chain activity creating a grid for classification. The row dimension is the decision tier and the column is the responsible unit (figure 2.1). Observe the decisions in each tier limit the options in the tiers below it.

The first decision tier, *strategic planning*, is typically driven by the lead time required for business planning, resource acquisition, and new product introduction. Here, decision makers are concerned with a set of problems that are three months to seven years into the future. Issues considered include, but are not limited to, what markets the firm will be in, general availability of equipment and skills, major process changes, risk assessment of changes in demand for existing products, required or expected incremental improvements in the production or delivery process, and the lead times for adding additional equipment and skills.

The second tier, *tactical planning*, deals with problems the company faces in the next week to six months. Estimates are made of yields and cycle times (lead times), the likely profile of demand, productivity and reliability of equipment, etc. Decisions are made about scheduling releases into the manufacturing line or staffing levels. Delivery dates are estimated for orders or response times for various classes of patients is estimated. Deployment of equipment and staffing is adjusted. The order/release plan is generated or regenerated, and (customer-requested) reschedules are negotiated.

The third tier, *operational scheduling*, deals with the daily execution and achievement of a weekly plan. Shipments are made, patients receive treatments, customers are waited on, serviceability levels are measured, and recovery actions are taken. Optimal capacity consumption and product output are computed.

Tier “3.5” *straddles operational and real time response*. For example, a monitoring system might observe a lot has entered a “process time window” and its “urgency” to be assigned has “increased”. A process time window is a sequence of activities that must be accomplished within a certain time limit or the lot might need to be scrapped due to some type of contamination. A non-factory example would be the “triage” system that occurs regularly in an emergency room where a patient is place is one of four or five categories based on urgency. Although this decision

does not directly assign the patient to a health care provider, it has a strong influence over the type and urgency of the assignment.

The fourth tier, *real-time response system*, addresses the problems of the next hour to a few weeks by responding to conditions as they emerge in relevant time. Within the demand-supply network, relevant time response is often found in two areas: manufacturing dispatch (assign lots to tools) and order commitment (available to promise, or ATP).

Within manufacturing, the decisions made across the tiers are typically handled by groups with one of two responsibilities: maintaining an enterprise wide global view of the demand-supply network and ensuring that subunits (such as manufacturing location, vendor, warehouse, etc.) are operating efficiently. Ideally all planning would be centralized; in practice complexity precludes this. Capacity planning is a good example. At the enterprise level, capacity is modeled at some level of aggregation, typically viewing a key tool set as a single capacity point. At the factory level, each tool, or potentially each chamber in a tool, is modeled.

Demand Supply (DS)Network Planning, Scheduling, and Dispatch (PSD)			
Activity Areas and Decision Tiers			
		Enterprise Wide, global view - central planning	Enterprise Subunits (manufacturing, distribution, retail) factory planning
Decision Tiers	Tier 1 Strategic	Enterprise wide Central Plan once or twice a year for 2 to 5 year horizon at aggregate level with forecasted demand focused on business scenarios. Net result strategic direction established and financial commitments made	Capacity Analysis typically at tool family level and overall manpower to support forecasted demand, creation of production flow and capacity information for central plan, determining new production processes to introduce and estimated learning curve
	Tier 2 Tactical	Enterprise wide central planning weekly/biweekly/monthly > create demand statement (current orders, forecasts) > capture capacity, WIP, BOM, business policy > central planning engine to match assets with demand > estimate supply line linked to demand, early warning, production requirements, chase situations	Capacity (tools and manpower) analysis to gauge impact of changing product mix, identify challenges, review and modify deployment decisions and manufacturing engineering requirements, and create capacity constraint information for central planning and WIP status. Monitor tool level performance and take appropriate actions. Establish rules and metrics to set global lot importance - example, how many priority classes, algorithm to set lot importance within a class, limits on number of expedites.
	Tier 3 Operational "daily"	Enterprise Wide central planning reduced focus / what if > what if commit on large orders > what if on major asset change > status of key WIP and actions to take if needed > cross factory signals	Provide information to central plan and daily factory adjustments > establish target outs, due dates on lots > maintenance priorities > short term changes in deployment > review key lot status and change priority (up or down) based on progress (either manually or dynamically) > one time changes in lot importance guidance > establish mfg lot vs development lot preference > revised projected outs for enterprise planning
	Tier 3.5 sub daily guidance	Change in Priorities, updated supply projections based on updated WIP or capacity status; change in customer reserved supply	As needed Updates to Guidance to support response decisions > regular updates to lot status based its progress, entering a time process window, status of short term manufacturing targets, WIP position and tool status > regular updates to tool status based on manufacturing engineering requirements, tool events, etc
	Tier 4 - Response	Available to Promise or Automated Order Commit process, cross factory signals	Dispatch Scheduling & Tool Response > assign sequence of lots to a tool > change status of a lot (for example on or off hold) > monitor signals from tools and respond as needed

Figure 2.1: Decision Grid for Demand Supply Networks

2.1 Basics of Enterprise Wide Central Planning

This activity involves the coordination of supply and demand (current and future) [17, 75] across each unit in the enterprise to determine which customer commitments can be made and what opportunities to chase. It consists of these core activities;

1. Create a demand statement
2. Capture the flow of materials in the demand supply network
3. Gather and collect key information from the factory

- 3.1. Project the completion of WIP to a decision point (often completion of the part).
- 3.2. a statement of capacity required and available
- 3.3. a statement of lead time or cycle time to complete a new start
4. Create a model that captures key enterprise relationships (Central Planning Engine – CPE)
5. Create an enterprise wide central plan by matching current and future assets with current and future demand using the CPE to create a future projected state of the enterprise and the ability to soft peg the current position of the enterprise to the projected future position. Information from the model includes
 - 5.1. a projected supply linked with exit demand
 - 5.2. identification of at risk orders either to a commit date or request date
 - 5.3. Synchronization signals across the enterprise
 - 5.4. Capacity utilization levels
 - 5.5. Ability to trace each production & distribution activity that supports meeting a demand.

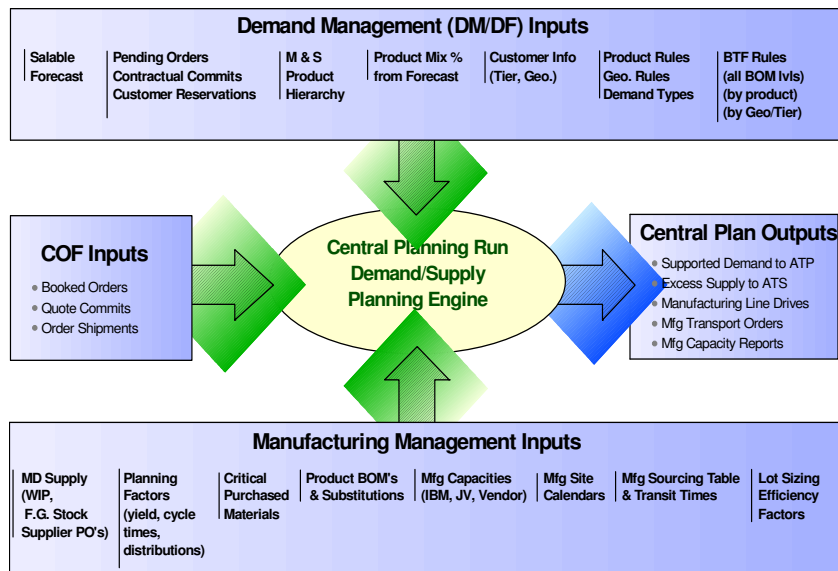


Figure 2-2: Typical Data Inputs and Outputs for Enterprise-wide Central Planning Engine

6. typically this is an iterative process done with different assumptions and conditions, for example
 - 6.1. different assumptions about capacity available
 - 6.2. different business policies for protective stock
 - 6.3. different commit dates and/or demand priorities for orders
7. execute the plan, that is,
 - 7.1. send signals to each core enterprise organization
 - 7.2. send projected supply to available to promise (ATP)

This flow is summarized in Figure 2-3.

Aggregate planning [1, 79, 87, 88] is typically focused on assessing the ability of the factory to meet certain demand looking to identify “broken” toolsets and creating the capacity inputs required by central planning model. Demand is typically stated as a starts profile and a lead (cycle) time commit for each part. For example, the factory has been asked to start 100 of Part A each day with a lead time of 20 days and start 50 of Part B with a lead time of 10 days. Historically, the focus is typically to identify “broken” tool sets (groups of tools current without sufficient capacity to meet this load (starts) and cycle time). Once these tools sets are identified there are two options: actions are taken to add capacity or central planning modifies the load profile. Typically (but not always) this is a slow and reasonably static process compared to the more dynamic central planning process. It operates more like an MRP (which simply tells the central planner what needs to be done to achieve a give demand), then a “best can do” model (which searches a wide set of options to best meet demand without violating constraints). That said, there is strong interest and need to upgrade this process to dynamically handle such factors as demand priority, cycle time versus load trade-off, and partially shared operations between tools. The basic steps are

1. Capturing representative product routes- sequence of operations, raw process time at the operation, and tool set consumed for each key operation.
2. Capturing a specific factory load – typically given in starts and lead time or cycle time commit.
3. Tool set characteristics: number of tools; tool availability;
 - 3.1. which operations the tool set handles, overlapping tool sets (partially shared operations between toolsets called a cascade; defined next section and [1]);
 - 3.2. its operating curve (Appendix 1, [2, 4, 19,38, 39])
 - 3.3. which establishes the trade-off between cycle time and tool utilization / capacity available;
 - 3.4. and reentrant flow (when a lot visits the same tool set multiple times, defined section 3.1 on wafer fabrication).
4. Allocation of tool sets to parts or families either as user input or based on history
5. A model that captures key relationships
6. Executing the model to determine how this load impacts toolsets
 - 6.1. Required utilization levels for each tool set to meet demand stated as starts including identifying broken tools (tools without sufficient capacity to meet demand)
 - 6.2. Some details on which parts are consuming how much of a tool set
 - 6.3. Capacity loss points (planned maintenance, high raw process time, etc.)
7. Providing information for central planning
 - 7.1. High level statement of capacity based on starts or a few key tool sets fixed for the specified cycle time to limit demand on the *factory mapping into the simple linear method to represent capacity consumption and capacity available*
 - 7.2. How additional capacity enables the factory to handle more starts or lower cycle time

Why is this different then central planning? Aggregate factory planning deals with

- the actual route (sequence of manufacturing actions to produce a finished part)
- the operating curve which establishes the trade-off between cycle time and tool utilization / capacity available.
- Partially Shared Operations (PSO) or Cascade which refers to different toolsets that overlap in the manufacturing operations they service, but the actions are not identical.
- Deployment Variability (described in section 2.2.2)
- Product mix transitions
- reentrant flow (described in section 3)
- determining a method to summarize capacity (required and available) to fit the traditional

simpler approach to handling capacity in central planning

All but the first item is described in more detail in later sections and make up much of the core of the ‘ongoing challenges’.

2.2.1.1 Partially Shared Operations (PSO) – Cascading

In **Table 2.1** we have three tools (A, B, and C) to service seven operations (001 – 007) creating a 7 by 3 grid. A 1 indicates that this tool can service this operation and 0 it can not. Assume there are no other tools that can service these operations and there are no other operations that these tools can service. This is called PSO or Cascade group since all of there are no tool/operation subsets that can be found that are independent – at some level each of these operations and tools are connected. From a tool perspective

- Tool A and Tool B “share” operations 1 and 2
- Tool A and C do not share any operations
- Tool B and Tool C share operations 3 and 5
- Therefore the connection between Tool A and Tool B is a second order link.

Table 2.1: Deployment Information for PSO Group				
	Tool A	Tool B	Tool C	no tools covering oper
oper001	1	1	0	2
oper002	1	1	0	2
oper003	0	1	1	2
oper004	0	0	1	1
oper005	0	1	1	2
oper006	1	0	0	1
oper007	1	0	0	1
number ops tool covers	4	4	3	

Table 2.2 has the same tools and operations as table 2.1, but with a different deployment allocation. In this situation there is NO overlap in operations between Tool B and Tool C, therefore there are two PSO groups. Tool A and Tool B (and the operations they service – 1, 2, 5, 6, and 7) and Tool C (operations 3 and 4).

Table 2.2: Deployment Information for 2 PSO Groups				
	Tool A	Tool B	Tool C	no tools covering oper
oper001	1	1	0	2
oper002	1	1	0	2
oper003	0	0	1	1
oper004	0	0	1	1
oper005	0	1	0	1
oper006	1	0	0	1
oper007	1	0	0	1
number ops tool covers	4	3	2	

An additional wrinkle to this feature is the raw process times for the same operation may differ between tools (Table 2.3).

Table 2.3: RPT information			
	Tool A	Tool B	Tool C
oper001	4	20	5
oper002	15	20	6
oper003	10	15	8
oper004	10	9	20
oper005	5	5	5
oper006	8	10	10
oper007	10	10	10

2.2.2 Deployment or Near Term Tool Planning

This decision is essentially determining which cells in table 2.1 get a 1 and which get a 0. Deployment) refers to determining which operations each tool will be qualified to handle over the short term. Typically a tool can service many operations, but a factory will limit the number of operations it is “allowed” to service to

- focus tools on the operations they are best suited to handle (speed, yield, etc)
- ensure a tool remains qualified to handle an operation by regularly servicing this operation
- tool specific restrictions which may limit a tool from serving some operations based on the operations it can currently service
- reduce work load on manufacturing engineering to maintain tool qualification
- make dispatching simpler by reducing options

2.2.3 WIP Projection

All central planning engines require each factory to project when current WIP in the line will reach either completion or staging point – typically called projected finish. The typical process involves capturing the remaining steps (route) until the lot reaches its end point and then making an estimate of how long the lot will need to finish these steps.

2.3 Factory Floor Dispatch

Dispatch refers to assigning a lot to a tool and requires balancing effective tool utilization with stable delivery (either to the commit date on the lot or to the number of exit lots per day or week). Typically there are different camps with substantially different views on how to make this decision. For example, manufacturing is looking to maximize output while the business team is as concerned about the lots for key clients. As we will see in a future section, what looks easy on the surface gets complicated quickly.

3. Basics of the Production of Semiconductor Based Package Goods

Semiconductor manufacturing is a complex process involving everything from growing silicon crystals, manufacturing silicon wafers upon which integrated circuits (ICs) are built, to the actual placement and soldering of chips to a printed circuit board ([1, 14, 49, 50]). Typically, the core manufacturing flow is wafers to devices to modules to cards (Figure 3-1).

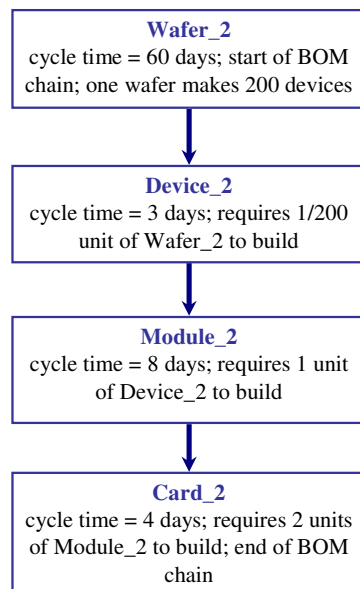


Figure 3-1: Simple Flow for Production of Semiconductor Based Package Parts

3.1 Wafer Fabrication

A wafer is a thin, round slice of a semiconductor material, usually silicon. These wafers are cut from a silicon ingot. This ingot is grown from a single crystal of silicon into a long cylinder. The goal of this process is to build hundreds to thousands of identical interconnected circuits (ICs) patterns (chips) on the wafer surface according to a specific circuit design. Once the wafer is cut, circuit components are sequentially built on the wafer surface. This is referred to as the front end. These components are then connected to form the interconnected circuits (called the back end).

The front end process is broken down into five different steps that are repeated many times. These steps include chemical clean and oxidation, photolithography, etching, ion implantation, and metal/insulator deposition. Chemical clean and oxidation prepare the wafer for photolithography. Photolithography is the process of “developing” the photoresist layer

(photosensitive material) to expose the circuit pattern by protecting the deposited material. The mask or reticle used by the photo tools controls the exposure process in a way that corresponds to circuit structures. Etching is the process of using chemicals to remove the unprotected materials in the photolithography step. Ion implementation modifies the wafer's conductive properties to complete the building of circuit components. Deposition is the process of depositing materials on the wafer surface to complete the transistor. These five steps are repeated as many times as the design requires, resulting in a three-dimensional, layered structure on the two-dimensional wafer surface. The number of layers varies considerably.

The back end fabrication process is an iterative set of “metallization activities” to wire or connect the components. The individual devices are interconnected by a system of fine metal lines and vias (vertical conducting plugs between metals on different layers) using a series of photoresist-patterned metal and insulator deposition and etching steps. The number of metal interconnect layers required depends on the design and also varies considerably. The flow in the back end is similar to the front end with “metallization” substituting for “ion implant”.

Following the patterning of the last metal layer, the wafer is covered by a final dielectric layer for passivation with openings etched in this film for making electrical contacts. The wafer is then diced into individual chips, each of which is then assembled into a package or module that provides for the attachment of wire bonds to the chip.

From a PSD perspective the critical insight is the repetitive use of the same core processes across the front end and back end and hence the same core equipment sets, is called “reentrant flow.” This generates the complexity of partially shared operations (PSO or cascading) and deployment decisions. The following example will illustrate these issues. **Figure 3-2** has a simplified set of major manufacturing activities to build one circuit layer.

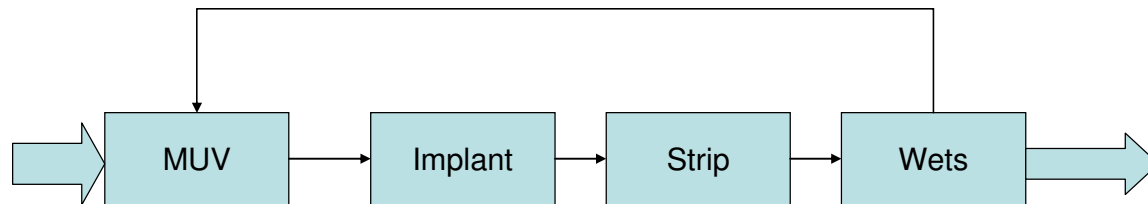


Figure 3-2 typical steps in building a circuit layer

This core set of activities is repeated multiple times to build a part and the number of iterations is different between parts. **Figure 3-3** shows the number of iterations through these same steps for Product A (three passes) and Product B (two passes)

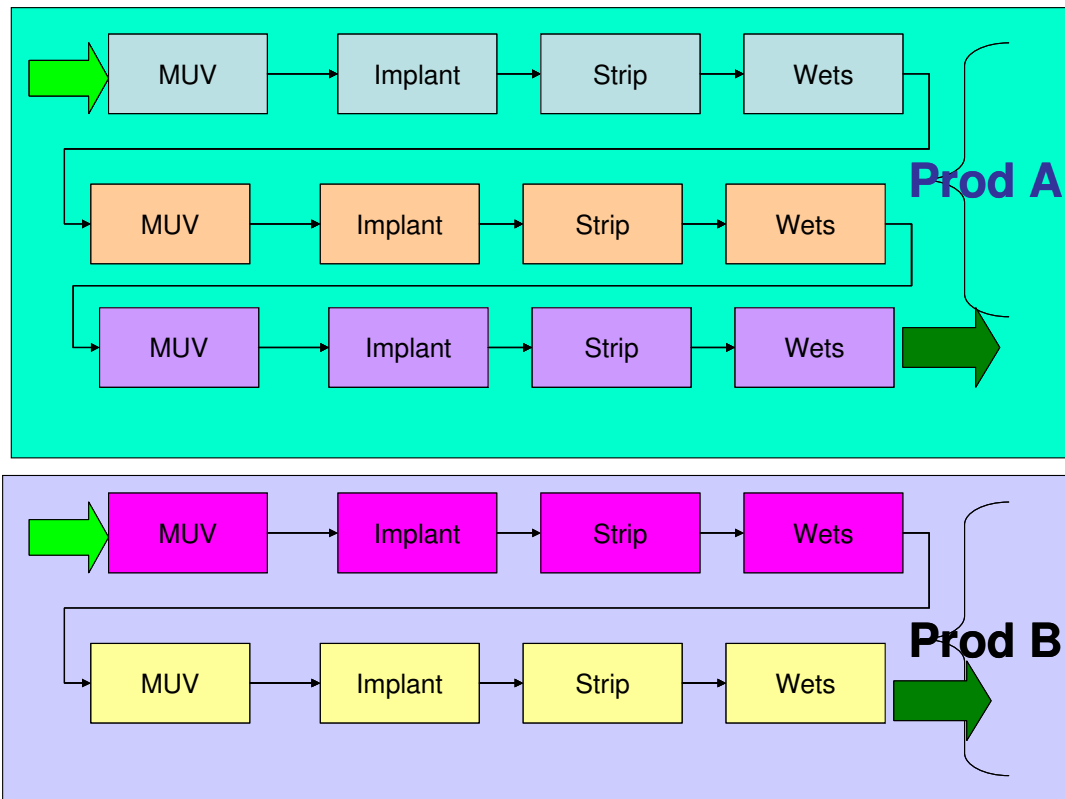


Figure 3-3: Example of reentrant flow for typical circuit construction

In this example Product A requires three passes through the MUV toolset and Product B two passes. Assume MUV is serviced by three tools (A, B, and C). Typically not all tools handle all operations (passes). **Figure 3-4** demonstrates the concept of partially shared tools sets for related, but different manufacturing operations. In this example there three tools (A, B, and C) to handle the MUV step. But not all tools can handle all steps. This is the deployment decision.

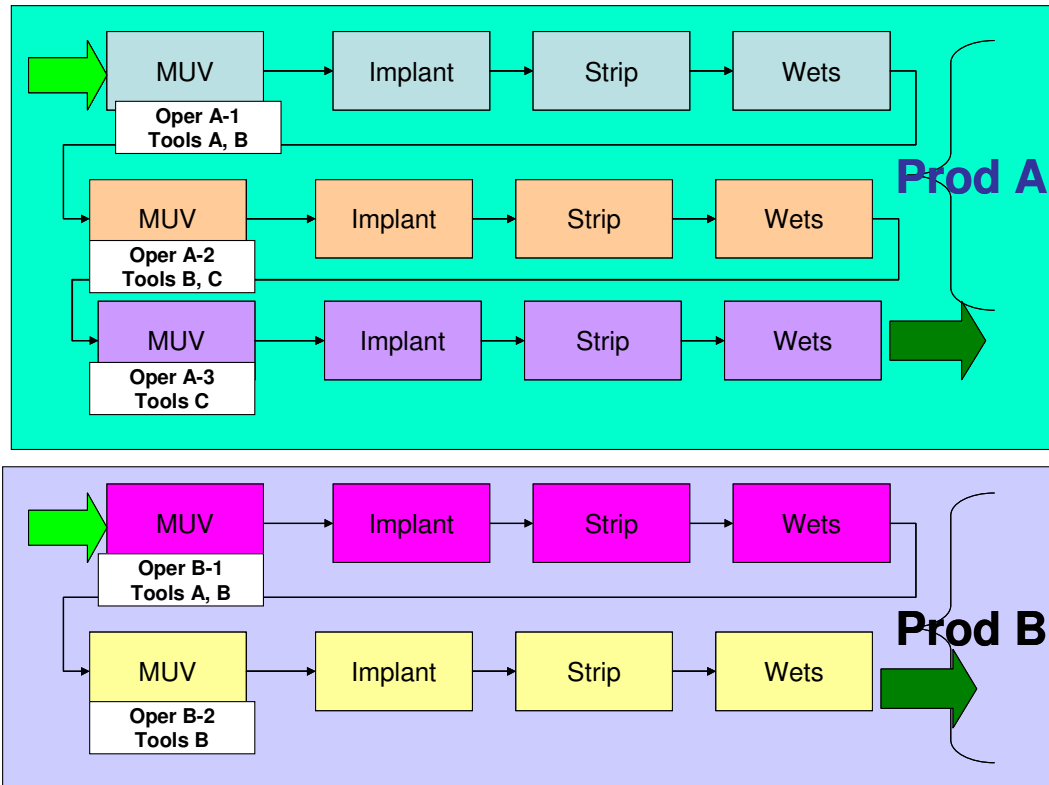


Figure 3-4 Example reentrant flow with Partial Shared Operations (PSO) or Cascading

In table form, the deployment decision would be represented as follows

Table 3.1: Deployment Information for Circuit Layers				
	Tool A	Tool B	Tool C	no tools covering oper
oper A-1	1	1	0	2
Oper A-2	0	1	1	2
Oper A-3	0	0	1	1
Oper B-1	1	1	0	2
Oper B-2	0	1	0	1
number ops tool covers	2	4	2	

The decision about which tools can handle which operations (deployment decision) is essentially determining which cells in table 3.1 get a 1 and which get a 0. Typically a tool can service many operations, but a factory will limit the number of operations it is “allowed” to service to

- focus tools on the operations they are best suited to handle (speed, yield, etc)
- ensure a tool remains qualified to handle an operation by regularly servicing this operation

- tool specific restrictions which may limit a tool from serving some operations based on the operations it can currently service
- reduce work load on manufacturing engineering to maintain tool qualification
- make dispatching simpler by reducing options

For complex designs, it may take three to six hundred individual steps to complete the entire IC fabrication process. Cycle times range from 30 to 130 days, and yield (percentage of quality wafers) can show significant variation. There is also wide variation in the demand patterns for wafers. A wafer that is a component of a video game has large demands over a 6-to-12-month period. In contrast, usually wafers that support specialty products have sporadic demands and those used in sensing and testing devices have stable demands. Those involved in computers or control units have moderate demands over a reasonable period of time and then fall into an end of life phase that must cover warranty issues. Demand is stated either as a finished wafer or a planned start. For all of the complexity and uncertainty in wafer fabrication, all enterprise planning engines have a simple representation of this manufacturing process: a few key capacity points and a limited number of purely serial decision points.

For additional details of the dispatch and scheduling complexity in wafer fabrication [43, 49]

3.2 Wafer Testing and Device Generation (Binning)

Once the circuits have been built on the wafers, they are tested to determine the resultant yield of operational circuits (good or bad, speed, power consumption, etc.) and tagged for reference. The wafers are then diced (cut) into individual units and sorted or binned based on the prior testing creating what is referred to as the device or “die bank.” This process is generally referred to as “binning” (illustrated in Figure 3-5). Observe it is the device (not the wafer) that is packaged and placed into video games, cell phones, laptops, etc. A single wafer generates anywhere from 30 to 900 devices, and sometimes, the exact type of device is not determined until it is tested. That is, the testing not only determines whether the device is usable or not, but also its final part identity. Differences between devices from the same wafer typically occur as result of speed and power consumption. Wafer testing has a short cycle time (3 to 10 days) and involves a purely sequential process through test operations except for rework.

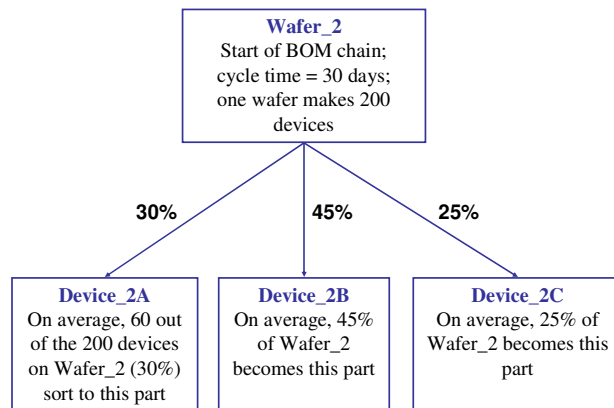


Figure 3-5: Simple Binning or Sorting Situation from Wafer to Device

3.3 Modules

The device step is followed by a sequence of assembly steps to mount devices onto a substrate (called wire bonding) and package to make a module. Packaging protects the fragile device inside and makes it suitable to be incorporated in other electronic products. These modules are further tested to determine electromagnetic and thermal characteristics. As with devices, in some cases the final identity of the part may not be determined until the test activities are done. Again, the cycle times are short compared to fabrication times.

For some modules, a significant amount of complexity is introduced when the production process requires a sequence of packaging and testing steps and/or the same module can be made from different processes that may or may not consume the same component part (a.k.a. alternate build paths). **Figure 3-6** provides an example of this type of complexity.

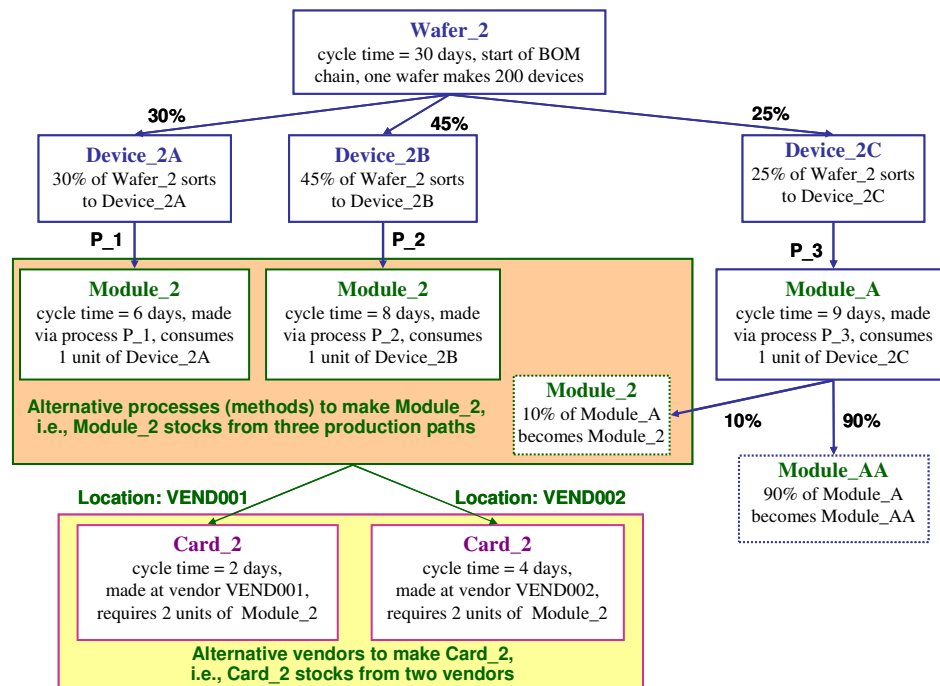


Figure 3-6: Alternative Processes to Make Module_2, Module Binning, and Card_2

3.4 Cards

The modules are eventually combined onto printed circuit boards to make cards. Finally, the cards are tested and those that pass are used in the assembly of a wide range of electronic products such as PCs, printers, and CD players. The cards are often produced at multiple vendor locations (**Figure 3-6**).

3.5 Total Journey

To create manageable enterprise wide supply chain models, we introduce some abstraction from the complex routing of lots through the production process. We project the continuous process onto a discrete set of part numbers (PNs). We use a bill of material (BOM) that specifies the component(s) of each PN to generate a graphical representation of the components needed for finished products, the alternative paths to produce the same PN, and the binning or sorting activities that determine the final status of a PN (**Figure 3-7**).

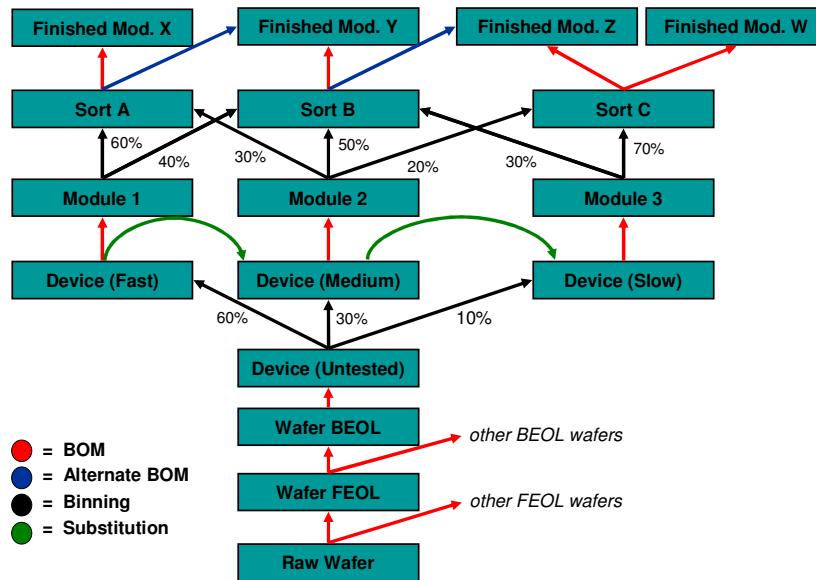


Figure 3-7: Graphical Representation of a Bill of Material Structure

Observe the variety (complexity) inherent within the total “journey from sand to module or card.” The first stage (fabrication of wafers and then testing and sorting into devices) has a long lead or cycle time where the capital expense for equipment ranges from hundreds of millions to billions of dollars. There is no assembly at this stage; in fact, the number of PNs goes from a few to many. In stage two (modules) and three (cards), the cycle times are short and the process is assembly and test. Typically, more than one component is needed in the assembly process and there are multiple ways to produce the same part. The number of finished goods explodes compared to the original number of wafer parts. A manufacturing facility typically specializes in only part of the process. As an example of a common product flow: the wafer is produced at location A; it is tested and diced at location B; the module is assembled at location C1, C2, or C3; it is tested at location D1 and then sent to location E1 or E2 to be assembled onto a card or sent directly to a customer. Within this network, different PNs that support a wide range of customers are constantly competing for the same tool or machine capacity.

The complexity extends to variability across time as well. Most of the planning parameters and business preferences are date or time effective. This includes cycle times, yields, capacity required, capacity available, binning percentages, manufacturing processes, and so on. For example, in Figure 3-6 the cycle time to produce Module_2 might reduce from 8 to 5 days in three weeks, and the entire process might be eliminated in 30 weeks.

To add to the complexity, **customer or exit demand may occur at any level** (card, module, device, wafer tested, and wafer untested), or be stated as required starts or allocation of capacity.

For example, a customer might place its primary demand as wafers and an additional “value add” demand for the firm to take some of the chips from the finished wafers and turn them into modules. In this case, the demand for finished wafers has a higher priority than the value add demand. If the firm doesn’t fully satisfy the value add demand, then the customer can have that work done by another vendor.

4 Central Planning Basics and Challenges

4.1 Overview of Best-Can-Do Enterprise Wide Central Planning Engines

The best-can-do (BCD) enterprise wide detailed central planning engine (CPE) is the control point for the flow of material or product within an organization, and focuses on how to best meet prioritized demand without violating temporal, asset (WIP and inventory), or capacity constraints. A CPE application minimizes prioritized demand tardiness and some aspects of cost, establishing a projected supply and synchronized targets for each element of the supply chain.

The core of the CPE process is matching assets with demand, which refers to aligning assets with demand in an intelligent manner to best meet demands. The alignment or match occurs across multiple facilities within the boundaries established by the manufacturing specifications, process flows, and business policies. Assets include, but are not limited to, starts (manufacturing releases), work in progress (WIP), inventory, purchases, and capacity (manufacturing equipment and manpower). Demands include, but are not limited to, firm orders, forecasted orders, and inventory buffer. The matching must take into account manufacturing/production specifications and business guidelines. Manufacturing specifications and process flows include, but are not limited to, build options, bill of material (BOM), yields, cycle times, anticipated date on which a unit of WIP will complete a certain stage of manufacturing (called a receipt date), capacity consumed, substitutability of one part for another (substitution), the determination of the actual part type after testing (called binning or sorting), and shipping times. Business guidelines include, but are not limited to, frozen zones (no change can be made on supplies requested), demand priorities, priority tradeoffs, preferred suppliers, and inventory policy. Many of the manufacturing specification and business guideline values will often change during the planning horizon (time effective).

The creation of a CPE plan requires a solver (sometimes referred to as a model or an engine) with the following core features:

1. method(s) to represent the (potential) material flows in production, business policies, constraints, demand priorities, current locations of asset, etc., and relate all this information to exit demand;
2. capture asset quantities and manufacturing specifications (parameters);
3. search mechanism(s) to generate a balanced supply chain plan, relate the outcome to demand, and modify the plan to improve the match;
4. display and explain the results of the best-can-do match.

The first task of any “best-can-do” CPE is to “flow material” and maintain a “feasible material flow” (see Graves et al 1995 for a review of material flow control mechanisms). Simply put, the CPE must decide a sequence of manufacturing starts to produce finished goods, and for each start the CPE places into the plan, the required component parts and capacity must be available and the manufacturing activity must be permitted on that day (e.g., it is not a shutdown day). For example, in [Figure 4-1](#) if the CPE decides to manufacture 10 cards on day 10 to be completed on day 14 to meet a customer’s demand, then on day 10 it must have 20 modules and tool/equipment capacity for the manufacturing process to consume. Typically, the CPE handles this requirement either implicitly with material balance equations or explicitly with explosion and implosion.

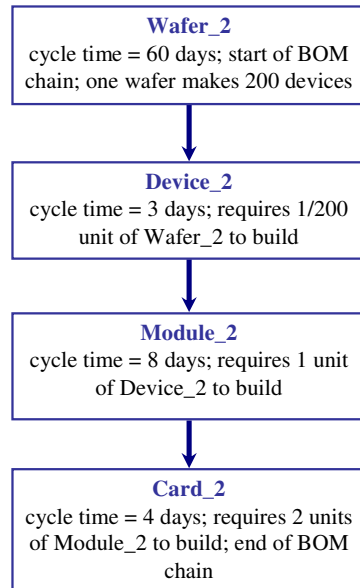


Figure 4-1: Simple Flow for Production of Semiconductor Based Package Parts

Explosion and implosion are the core processes of the CPE which either move work units (WIP or starts) forward (implosion) to project completed parts or backward (explosion) to determine starts required across the bill-of-material (BOM) supply chain following the appropriate manufacturing data such as cycle time, yield, capacity, and product structure. We typically use implosion to estimate what finished goods will be available to meet demand and explosion to estimate what starts are needed at what due dates to ensure meeting the existing demand on time.

To review implosion and explosion, consider Figure 4-1 again which represents a simple production flow. The first manufacturing activity is the production of Wafer_2. This manufacturing activity has a cycle time of 60 days, i.e., it takes on average 60 days to take a raw wafer and create a completed wafer with the part ID Wafer_2. The second activity is device production. Creating one unit of Device_2 requires 3 days and consumes 1/200th unit of Wafer_2. Module_2 consumes one unit of Device_2 and takes 8 days to produce. Finally, Card_2 consumes two units of Module_2 and takes 4 days to produce.

Referencing Figure 4-1, implosion can be illustrated with the following example. Manufacturing estimates that four units of Device_2 will be available or completed on day 10. This is called a projected receipt. If manufacturing immediately uses these four units to produce Module_2, on day 18 ($10 + \text{"Module\_2 cycle time"} = 10 + 8 = 18$) four units of Module_2 will be completed. Continuing the projection process, the four units of Module_2 are immediately used to create two units of Card_2, which will be available on day 22 ($18 + \text{"Card\_2 cycle time"} = 18 + 4 = 22$). The implosion process enables manufacturing to estimate the future supply of finished goods.

Again referencing Figure 4-1, explosion can be illustrated with an example. To meet demand for one unit of Card_2 on day 20, the plant must have two (completed) units of Module_2 available on day 16 ($20 - \text{"Card\_2 cycle time"} = 20 - 4 = 16$). This generates an exploded demand of two units of Module_2 with a due date of day 16. To continue the explosion process, to produce the two units of Module_2, the plant must have two units of Device_2 available on day 8 ($16 - \text{"Module\_2 cycle time"} = 16 - 8 = 8$). Next, the device demand is exploded creating a demand for 2/200th units of Wafer_2 on day 5 ($= 8 - 3$). This exploded information creates the guidelines for manufacturing to meet existing demand. For example, the device department must start production

of two units of Device_2 no later than day 5 to meet the demand for one unit of Card_2 on day 20. Since the cycle time to produce Wafer_2 is 60 days, it needs to have one already in production and close to completion.

Within the explosion and implosion process is a method called “demand pegging.” This method links each allocation of an asset or creation of a start with either a specific exit demand, or, at a minimum, the demand class or priority (relative importance of demand) associated with the exit demand being supported. Using the explosion example described above, if the exit demand for one unit of Card_2 on day 20 has a demand class of 3, each exploded demand will carry that demand class with it. Therefore, the units of Module_2 that are started on day 8 will have a demand class of 3. Similarly, if 3 units of Card_2 are desired on day 20 for a customer with demand class 5, then 6 units of Module_2 to be started on day 8 will also have a demand class of 5. The total required starts picture on day 8 is 8 (2+6), with 2 units with demand class 3 and 6 units with demand class 5. If by chance, there is only enough capacity on day 8 to start 2 units of Module_2, they will be allocated to the more important demand (demand class 3).

4.2 Challenges of Scope and Scale for Matching Assets with Demand in Central Planning

“The great 20th century revelation that complex systems can be generated by the relationships among simple components” [30] applies to supply chain planning (and almost all aspects of planning, scheduling, and dispatch).

Although simply creating a feasible central plan which maintains material balance, observes date effectivity, obeys business rules, captures existing WIP and inventory, and does a rough job at meeting demand is by itself challenging, it is no longer sufficient for a firm to remain competitive. The failure to “create a more accurate assessment of supply” forces the firm to compensate with slack (Galbraith 1973) or inefficiencies that leaves it at a competitive disadvantage. The purpose of this section is to identify and describe some of the key challenges a CPE must handle to provide an accurate assessment of supply. By “handle,” we mean provide a mathematical representation of the individual characteristics, the system or relationships, and a method to search for an intelligent, if not provably, optimal solution. In the following paragraphs, we describe the challenges of demand class; simple binning; complex binning, substitution, and alternative bill of materials (BSA); lot sizing; sourcing; fair share; and commit date versus request date.

Leachman [52], Kempf [50], and Denton [14] also provide excellent reviews of the complexities/challenges within the production of semiconductor based parts. Leachman focuses on binning and demand class; Kempf develops and explains the stochastic challenge (uncertainty in demand and supply); and Denton deals with demand class and lot sizing in detail.

4.2.1 Allocating Perishable and Non-Perishable Assets Based on Demand Class

A fundamental decision found in most aspects of planning and scheduling is deciding which demand “gets to go first”? That is, when more than one demand needs either capacity (a perishable asset) or a component part (typically viewed as a non-perishable asset) and there is not sufficient supply to meet all immediate demand, the question is which demands get the asset and which have to wait. The CPE needs to allocate the asset based on the relative importance of the demand (indicated by demand class or priority) and the impact on delivering the finished goods on time.

Figure 4-2 shows a simple example of allocating supply of part (non-perishable asset) to meet demand. Module_1 and Module_2 are both made from Device_12 with cycle time 10 and 4 days, respectively. The demand for Module_1 is 10 units on day 10 and 15 units on day 12. The demand for Module_2 is 8 units on day 5 and 2 units on day 6. The key decision for the CPE is

how to best allocate supply from Device_12 to the two modules (represented by the two boxes titled “Supply Amt” in Figure 4-2).

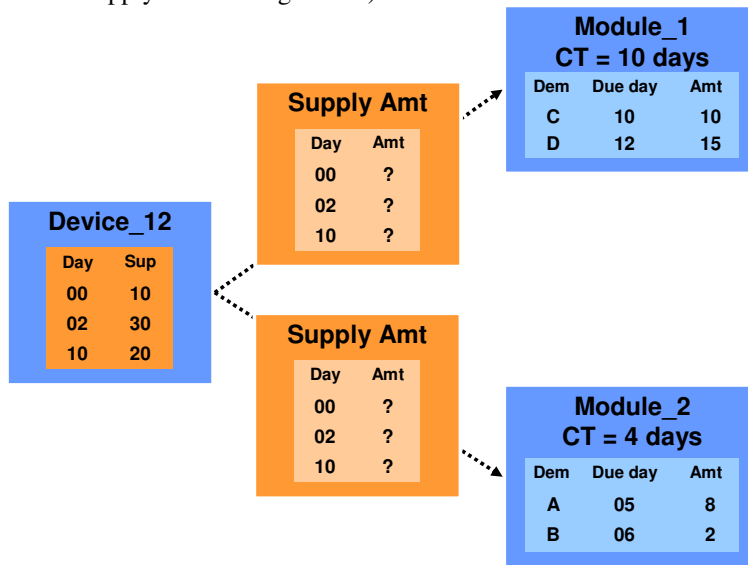


Figure 4-2: Example of Non-perishable Asset Allocation

One solution for the situation in Figure 4-2 might be: (1) immediately allocate 8 of the 10 units of Device_12 on hand to meet demand A (8 units of Module_2 on day 5) 1 day early on day 4 (=0+4); (2) immediately allocate the remaining 2 units of Device_12 on hand to meet demand B (2 units of Module_2 on day 6) 2 days early on day 4; (3) on day 2, allocate 10 units of the projected supply of 30 units of Device_12 to demand C (10 units of Module_1 on day 10) 2 days late on day 12 (=2+10); (4) on day 2, allocate 15 units of the projected supply of 30 units of Device_12 to demand D (15 units of Module_1 on day 12) on time (12=2+10). The score card for this solution is demand A early, demand B early, demand C late by 2 days, and demand D on time. Table 4-1 summarizes this solution.

Demand ID	Type	Commitment		actual delivery		Delta schedule
		date	Quantity	date	quantity	
A	Module_2	05	8	04	8	1
B	Module_2	06	2	04	2	2
C	Module_1	10	10	12	10	-2
D	Module_1	12	15	12	15	0

A second option could be: (1) 10 units of Device_12 are allocated to Module_1 on day 0 to cover demand C; (2) 15 units of Device_12 are allocated to Module_1 on day 2; and (3) 10 units of Device_12 are allocated to Module_2 on day 2. The score card for this solution is: demand A is met 1 day late, demand B is met on time, demand C is met on time, and demand D is also met on time. Table 4-2 summarizes this solution.

Demand	Type	commitment	actual delivery	delta

ID		date	quantity	date	quantity	schedule
A	Module_2	05	8	06	8	-1
B	Module_2	06	2	06	2	0
C	Module_1	10	10	10	10	0
D	Module_1	12	15	12	15	0

Which is better? If the four module demands are exit demands, the answer depends only on the relative importance of each demand and the business policy on “sharing the pain” if a demand cannot be met on time. If demand A is demand class 1 (the lower the value, the more important the demand) and demands C & D are demand class 3, the first solution is the logical choice. If the demand classes are reversed, the second solution is the logical choice. What if all of the demands have the same demand class? Do we go with solution two since demand A is just one day late? Do we meet part of demands C and A on time and the other part is late (can we split the order)? Or do we meet all of demand C and part of demand A on time? This typically depends on the business policy of the enterprise.

If the four module demands are not exit demands, then in addition to the demand class, we need to also assess whether meeting the module demand on time ensures meeting the exit demand on time. For example, if demand A goes into the exit demand for CARD01 for the XYZ customer and the board (modules go on boards to make a card) required is two weeks late, there is no point in worrying about meeting module demand A on time.

Observe the CPE needs three core functions to handle this type of decision: (1) being able to “peg” or pass down the priority of the exit demand to all intermediate demands, (2) a method to pass in and represent the business policy, and (3) mechanisms to search for options and assess the tradeoffs relative to overall efficiency and current business policy.

Figure 4.3 identifies the basics for allocating limited capacity-

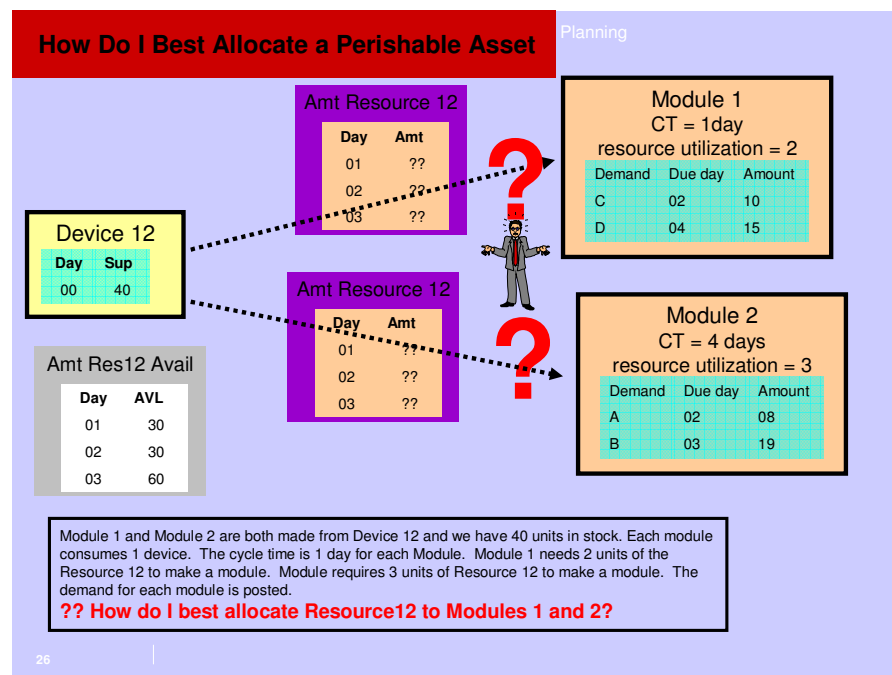


Figure 4.3: Allocating Limited Capacity

4.2.2 Simple Binning (or Sorting) with Downgrade Substitution

Simple binning with downgrade substitution refers to the classification of a part into one of a set of mutually exclusive and exhaustive categories based on some key performance factor using a series of testing procedures conducted after completing a set of manufacturing processes. In semiconductor manufacturing, the most common (but not the only) location for binning is the completion of wafer fabrication where hundreds of devices are cut from a single finished silicon wafer. Because of the random variation in the wafer fabrication process, working conditions for the devices on the same wafer vary. Therefore, before proceeding any further in the manufacturing process, devices have to be classified into different categories (each having a unique part number) after being cut from the finished wafer (typically today, the actual testing is done on the wafer). Clock speed, for example, is usually among a number of key performance factors to be tested for each device. Besides electronics, downgrade substitution can also be seen in such industries as consumer goods (bicycles) and building materials (grades of wood).

Figure 4-4 illustrates a typical binning scenario after the wafer fabrication process, where 50 percent of the time, the device is tested to have a “grade A” or top performance; 30 percent of the time, it has a “grade B” or medium performance; and 20 percent of the time, it has a “grade C” or low performance. These percentages are referred to as binning percentages and can generally be observed in semiconductor manufacturing as a result of binning testing. Another phenomenon commonly associated with binning is substitution between materials of the same type. Quite often, parts with a higher performance can substitute for parts with a lower performance if necessary. This form of material substitution is generally called downgrade substitution, and typically, it occurs when there is a shortage of the lower performance part accompanied by an overage of the higher performance part. In Figure 4-4, the dotted arrows indicate that “grade A” devices (Device_1) can substitute for both “grade B” devices (Device_2) and “grade C” devices (Device_3), and “grade B” devices can substitute for “grade C” devices. The binning percentages can and do change over time.

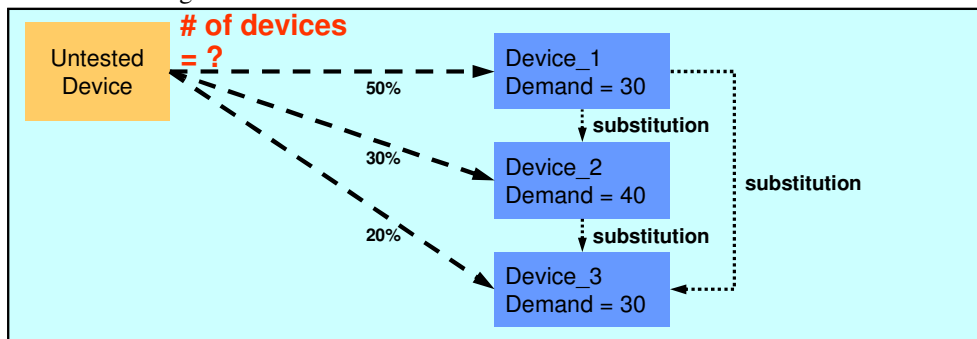


Figure 4-4: Simple Binning with Downgrade Substitution after Wafer Fabrication

The challenge is to make an optimal use of co-products and substitution to avoid overstating the number of wafer starts required to meet demand. If the demand is 30 for Device_1, 40 for Device_2, and 30 for Device_3 (Figure 4-4), the challenge is to determine the minimum number of wafers/devices that must be produced to meet all three demands. One simple rule is the maximum of the quantity required for each device divided by its binning percentage. Continuing the same example, we would need 150 devices, which equals maximum $(30/0.50, 40/0.30, 30/0.20) = \text{maximum}(60, 133, 150)$. As shown in Figure 4-5, such a rule will leave an excess inventory of 50 devices, with 45 contributed by Device_1 and 5 contributed by Device_2. If we optimally account for co-products and substitutions, the minimum number of devices required to meet all three demands is 100—testing 100 devices creates 50 of Device_1, 30 of Device_2, and 20 of

Device_3 (Figure 4-6). The extra 20 of Device_1 are used to cover the shortfall of 10 of both Device_2 and Device_3.

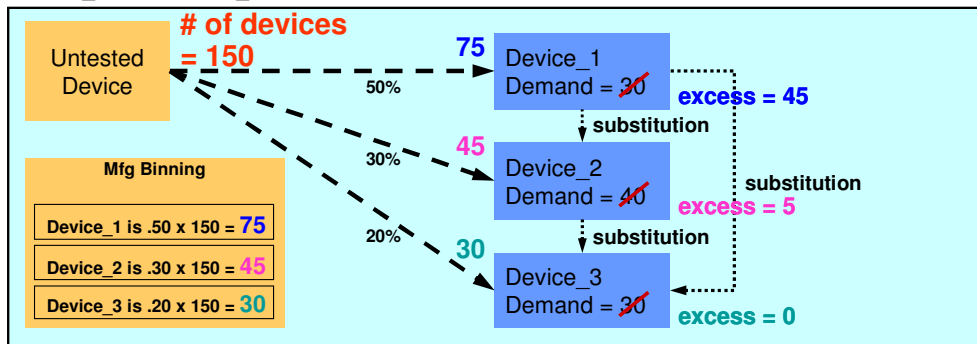


Figure 4-5: Maximum Quantity of Starts Leaves Excess Inventory of 50 Devices

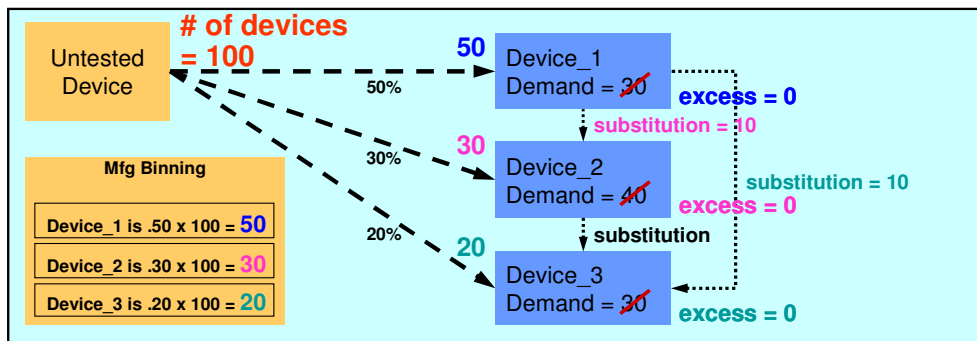


Figure 4-6: Optimal Number of Devices Meets All Demands and Leaves No Inventory

Other factors complicating the determination of the minimum number of starts required to meet demand in simple binning production structures include: demands for devices that spread throughout the planning horizon, existing inventory, projected WIP completion, and binning percentages and allowable substitutions that change during the planning horizon (date effectivity). Additionally, the central planning engine must locate and isolate these binning situations in a large, complex demand-supply network, as well as maintain full traceability and handle demand priorities (Figure 4-7).

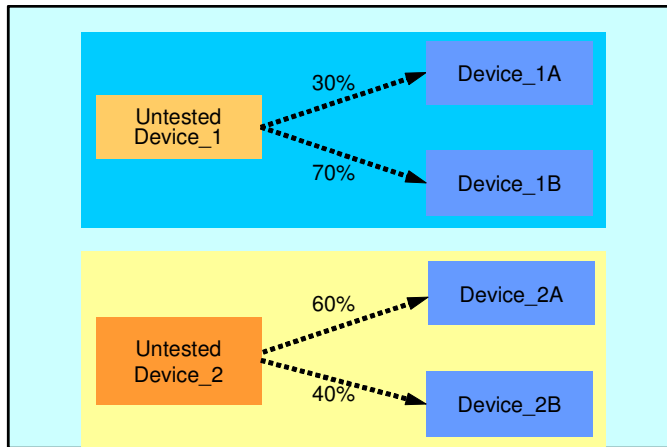


Figure 4-7: Two Binning Situations That Would Be Solved by Separate LPs

4.2.3 BSA: Complex Binning, General Substitution, and Alternative Bill of Materials (BOM)

Within the production of modules, an increasingly common manufacturing characteristic is alternative BOM (bill of material) structures, general substitution, and complex binning. In its simplest form, alternative BOM means two or more manufacturing processes are available to produce the same PN (part number). For example, Figure 4-8 illustrates a scenario in which process P1 and P2 can both be used to produce Module_9. If P1 is selected, the process will consume Device_8A; if P2 is selected, the process will consume Device_8B. Complex binning refers to a situation where one binning activity immediately invokes another, or substitutions are permitted across binning activities.

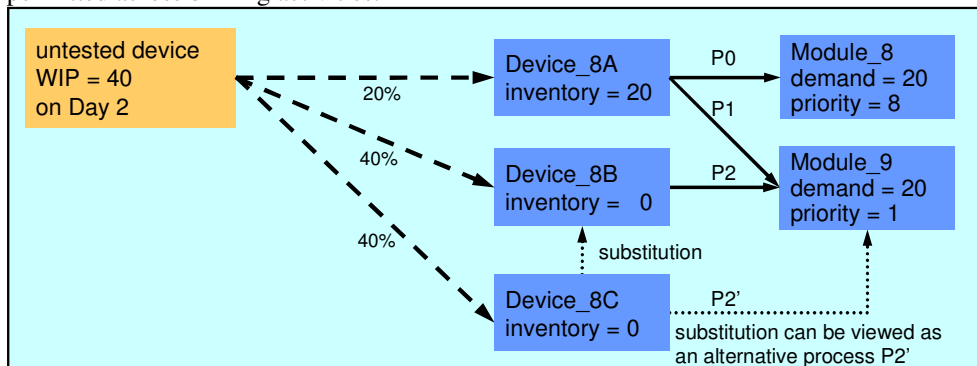


Figure 4-8: Illustration of Alternative BOM Structure with Substitution

To get a sense for the decision challenges created by alternative BOMs, we will use Figure 4-8 to look at just the “explosion” question within the CPE. Two processes (P1 and P2) can be used to build Module_9 where P1 consumes Device_8A and P2 consumes Device_8B. Additionally, Device_8C can generally be substituted for Device_8B, which means if Device_8B is not available and Device_8C is available, process P2 can use Device_8C to make Module_9. Conceptually, this general substitution can be viewed as the third alternative BOM option (call it P2-prime or P2’).

The explosion engine must determine how to explode demand for Module_9 and Module_8 back to the device level. Should it be half to P1 and half to P2, 2/3 to P1 and 1/3 to P2,

or all to P1? Should P2' be considered? The objective is to divide the demand for Module_9 across P1 and P2 (and perhaps P2') to best use the existing inventory and WIP, minimize new starts, and meet other relevant guidelines such as sharing percentages and capacity. Determining the best result requires an extensive search through the entire BOM structure.

In Figure 4-8, assume the priority for demand for Module_9 (which is 1) is higher than that for Module_8, and twenty units are ordered for both modules. A quick search reveals that twenty units of Device_8A are in inventory, which can be used to make Module_9 with process P1 or Module_8 with process P0. There is no current inventory for Device_8B or Device_8C to make Module_9 using process P2 or P2' (the substitution). Most heuristic based search engines that guide the explosion through alternative BOM structures would explode the 20 units of demand for Module_9 down the P1 process (or leg). This would consume all of the 20 units in inventory for Device_8A and leave the demand for Module_8 totally uncovered; assuming building every unit of module would consume one unit of device (Figure 4-9). To meet the demand for Module_8, a planning engine would require the demand-supply network to produce 100 new untested devices if the engine was not aware of the expected projection of the 40 untested devices (which could be tested into 8 units of Device_8A ($=40 \times 0.20$), 16 units of Device_8B ($=40 \times 0.40$), and 16 units of Device_8C ($=40 \times 0.40$)). On the other hand, if a planning engine is aware of this information, it would only require 60 new untested devices since there is a projection of 8 Device_8A ($60 = (20 - 8) / 0.20 = 12 / 0.20$).

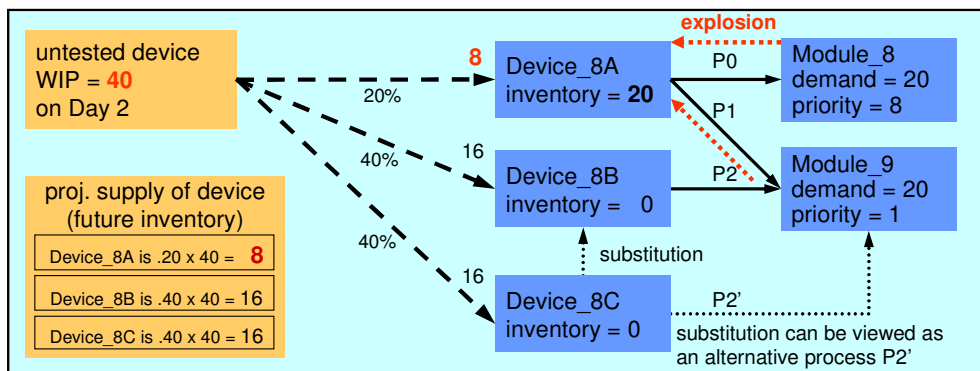


Figure4-9: Awareness of Asset Availability Affects New Starts

However, a broader search would uncover all available options at untested device and avoid the conflict for Device_8A between Module_8 and Module_9. There are 20 units of demand for Module_8 and it can only be made from Device_8A. So the question is: are there other options to meet the demand for Module_9? There are 40 units of projected WIP at the untested device and after binning, 8 will become Device_8A, 16 will become Device_8B, and 16 will become Device_8C. Since Device_8C can be substituted for Device_8B, there are actually 32 (16+16) future devices that can be used to produce Module_9 (but not Module_8), which is more than enough to meet the 20 units of demand for Module_9. So it is probably not optimal to explode the demand for Module_9 down the P1 leg. As illustrated in Figure 4-10, both demands can be met without using any new untested devices:

1. Assign the 20 units of inventory for Device_8A to be used to produce 20 units of Module_8 (blue dotted arrow).
2. Explode the 20 units of demand for Module_9 into a need for 20 units of Device_8B (red dotted arrow). Net this need against the 16 projected Device_8B, resulting in a need for 4 new Device_8B.
3. Use 4 out of the 16 projected Device_8C to meet the need for 4 new Device_8B.

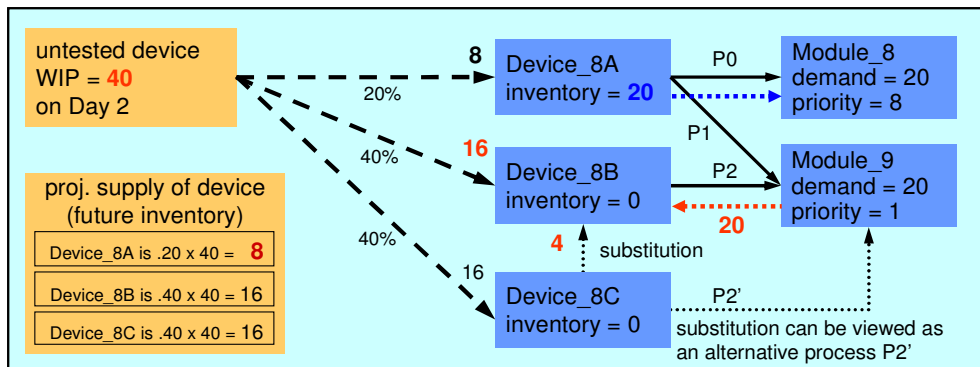


Figure 4-10: A Smarter Option—Meeting Both Demands without Using New Devices

Figure 4-11a and Figure 4-11b illustrates how the complexity can grow quickly and Figure 4-12 illustrates potential opportunities for parallelization through dynamic partitioning (indicated by different colors).

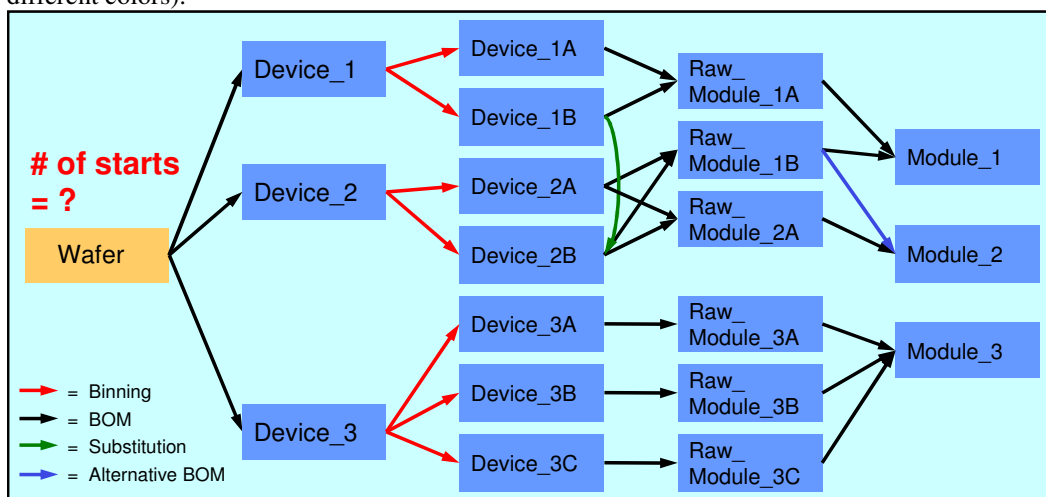


Figure 4-11a: Complexities across the Bill of Material Structure

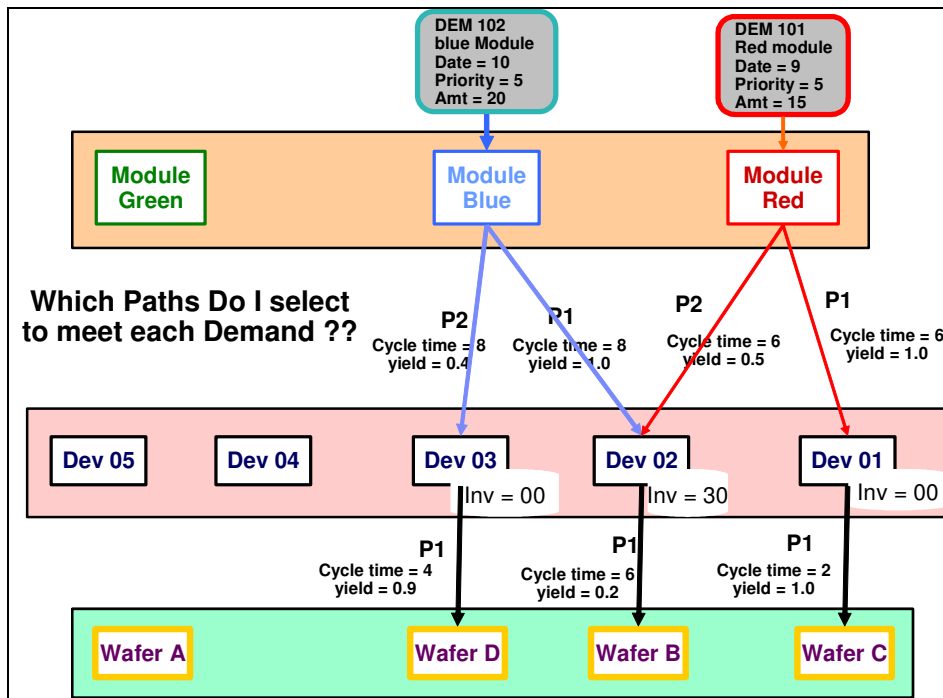


Figure 4-11b: Alternative Paths to Meet Demands that Share Common Assets

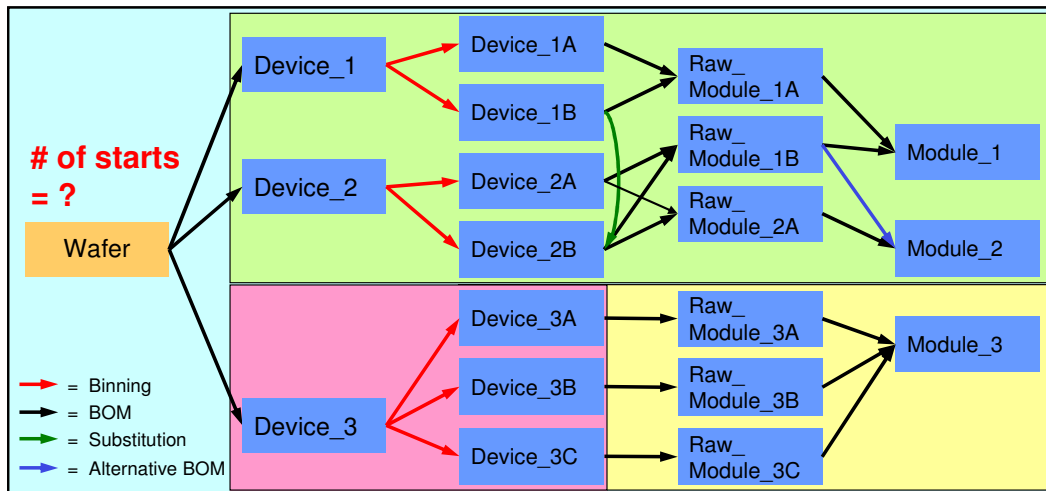


Figure 4-12: Opportunities for Parallelization through Natural Partitions

Increasingly, the CPE and the planner have to handle all three complexities: alternative BOM, general substitution, and complex binning. Figure 4-13 illustrates this “bundling.”

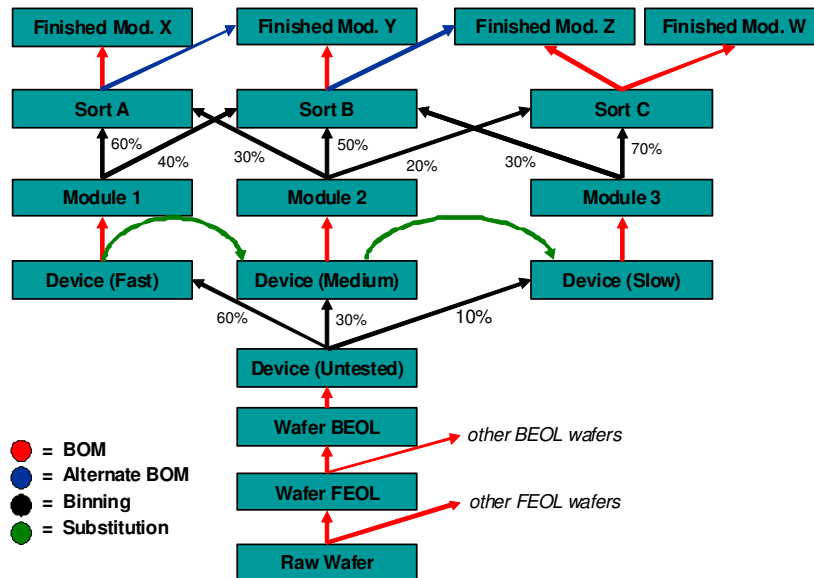


Figure 4-13: Example of Bundling of Complexities

4.2.4 Lot Sizing

Lot sizing refers to a core manufacturing characteristic that the number of units in each lot of activity can have a significant impact on productivity. This preference/requirement is typically described by minimum lot size, maximum lot size, and multiples. Probably the easiest and most common example is minimum lot sizing.

Within a manufacturing facility, the same tool set is used to process a variety of products. For example, the same testing equipment is used to investigate and sort a wide variety of modules. However, there is a substantial “setup cost” when the tool set switches from one part to another. Therefore, the manufacturing unit wants to get a “return” on this setup cost by processing a minimum quantity of each part. It does not want to do 3 of part A, then 5 of part B, then 2 of Part C, etc.; rather, it wants to do 30,000 of part A, then 5,000 of part B, then 20,000 of part C, etc.

This requirement creates a set of challenges for the CPE. First, the engine has to accommodate that in some cases the manufacturing release number can only come in discrete quantities (challenge for linear programming). Second, the engine needs a mechanism to keep track that the same lot will be accommodating different demand priorities. Third, not all lot sizing requirements are firm: the manufacturing facility may prefer to process 30,000 of part A instead of 3, but it can do just 3 if it is important enough.

4.2.5 Sourcing

If an enterprise has more than one supply location which provides certain parts, it typically wants to maintain some type of balance between the workload on each facility. This “balance” is typically described with a reasonably complex set of business rules. Often, these rules arise from complex contractual obligations to suppliers that vary over time and business conditions.

4.2.6 Fair Share

Fair share refers to sharing limited supply or capacity among a set of equally important customers, as opposed to filling their orders in some random fashion. For example, A and B are the firm's most important customers, and they both have just requested 100,000 units of the same part to be delivered on the same day (200,000 units in total). Suppose the available supply is only 140,000 units, 60,000 short of the combined demand quantity. If no concern is given to the final delivery quantity, the CPE may generate a supply chain plan which ships 100,000 units to A (i.e., A's order 100% filled) and 40,000 units to B (i.e., B's order only 40% filled) despite the fact that they are equally important. With fair share, on the other hand, both customers will have 70% of their order delivered (and share the same degree of "pain"). The importance that all customers with the same level of importance to the company receive the same level of service is quite obvious. This requirement creates an additional challenge for the CPE.

4.2.7 Customer Request versus Customer Commit

Customer request versus customer commit (date, quantity, priority) is emerging as a key complexity directly impacting customer satisfaction. Typically, a customer requests to receive a certain number of part(s) on a certain date and this is the customer request date and request quantity. Next, the enterprise reviews the request and responds with a date and quantity to which it will commit which becomes the commit date and commit quantity. The customer and supplier may iterate a few times on date and quantity until a firm commit date and quantity are established. The supplier often associates some priority with the firm demand. Historically, this is the only date, quantity, and priority the central planning engine has for each demand, that is, without using the original request date and quantity information during its solution process. In current practice, at best some post processing activity occurs to identify "low hanging" opportunities to meet a few key request dates.

To start, let's consider the simple example illustrated in Figure 4-14 where there is a demand for 100 units of part A with a request date of 06/01 and a commit date of 07/01. Assume WIP (work-in-progress) exists projected to come to stock on 06/01. A two-pass solution could do the following: in the first pass, WIP is used to cover the 07/01 commit date and then netted out and became unavailable for the second pass. In the second pass, no unnetted asset remains to cover the 06/01 request date. The two-pass process did not pick up the possibility that WIP could cover the request date. A solution that carried both the commit and request date would recognize the WIP could be used to make the request date with the primary search process.

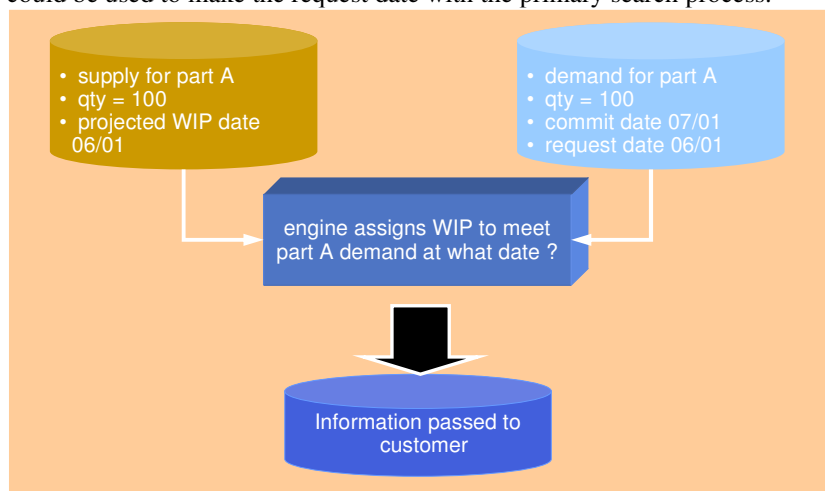


Figure 4-14: Simple Example of Handling Commit and Request Dates

A more complicated example is shown in **Figure 4-15**. Here we have 5 demands for part XYZ and anticipated supplies arriving on days 2, 4, and 5. All demands have a commit date **but only two (B and E) have a request date**. To keep the example simple, we assume all demands have the same priority. The key business question is what date can we deliver the product to the customer! Can we meet the commit dates? Can we meet any of the request dates?

Anticipated Supply			Demands for Part XYZ			
supply ID	date	amount	order id	commit date	request date	amount
SUP01	2	30	A	2	none	15
SUP02	4	10	B	3	2	5
SUP03	5	20	C	4	none	10
			D	5	none	20
			E	6	2	10
total		60	total			60

Figure 4-15: Demand and Supply Information for Part XYZ

In **Figure 4-16**, we show a typical solution focused only on the commit date—all the commit dates but none of the request dates are met. The on-time delivery (OTD) score for the commit date is 0, but the OTD score for the request date is -5.

Demands for Part XYZ				Allocated Supply to meet Demand				on time delivery	
order id	commit date	request date	amount	supply ID	date	amount	date to customer	for commit	for request
A	2	none	15	SUP01	2	15	2	0	NA
B	3	2	5	SUP01	2	5	3	0	-1
C	4	none	10	SUP01	2	10	4	0	NA
D	5	none	20	SUP03	5	20	5	0	NA
E	6	2	10	SUP02	4	10	6	0	-4
total			60	total		60		0	-5

Figure 4-16: Solution Focused Only on Commit Date

Sometimes, a simple post processing routine is executed to identify “easy” opportunities to meet request date. In our example, the supply used to meet demand B has an anticipated supply date of 2, therefore the customer can have the supply on day 2. The results of a post processing routine are shown in **Figure 4-17**. With this routine, we are able to meet the request date for order B but not the request date for order E.

Demands for Part XYZ				Allocated Supply to meet Demand				on time delivery	
order id	commit date	request date	amount	supply ID	date	amount	date to customer	for commit	for request
A	2	none	15	SUP01	2	15	2	0	NA
B	3	2	5	SUP01	2	5	2	1	0
C	4	none	10	SUP01	2	10	4	0	NA
D	5	none	20	SUP03	5	20	5	0	NA
E	6	2	10	SUP02	4	10	6	0	-4
total			60	total		60		1	-4

Figure 4-17: Solution Obtained with a Simple Post Processing Routines

Figure 4-18 shows the results of carrying both commit date and request date within the solution process. Both request dates (demand B and E) are met. This is accomplished by swapping SUP01 and SUP02 between demand C and E.

Demands for Part XYZ				Allocated Supply to meet Demand				on time delivery	
order id	commit date	request date	amount	supply ID	date	amount	date to customer	for commit	for request
A	2	none	15	SUP01	2	15	2	0	NA
B	3	2	5	SUP01	2	5	2	1	0
C	4	none	10	SUP02	4	10	4	0	NA
D	5	none	20	SUP03	5	20	5	0	NA
E	6	2	10	SUP01	2	10	2	4	0
total			60	total		60		5	0

Figure 4-18: Solution in Which Both Request Dates Are Met

The challenge is to carry both commit and request information (date, quantity, priority, etc.) intrinsically within the CPE and identify opportunities to come closer to customer requests (respecting their relative importance) without sacrificing any commitments which have been made. This requires carrying multiple dates and quantities throughout the explosion component of the CPE, identifying opportunities to assign assets to exploded demand to improve the posture of the exit demand, and being robust enough to handle such complexities as binning, substitution, and lot sizing.

4.2.8 Minimum Starts

Sometimes, a planner will want the CPE to ensure that a minimum number of starts occur at a certain point in the production process. For example, in Figure 4-1, the planner may want a minimum of 100 Module_2 per week over the next 20 weeks to smooth the manufacturing flow or to meet contractual agreements (e.g., a vendor may be willing to promise a shorter cycle time if it has a “smoothed” start plan). In these cases, the CPE must explode the exit demand back to the identified minimum start manufacturing activity, compare the minimum start value with the required value, then continue the explosion process. On the return trip (implosion), the CPE must also adapt for the relative priority of the minimum start.

4.9 Date Effective Parameters, Substitution Rules, Bill of Material, Capacity, etc.

“Date effective parameters” are a challenge that is often underestimated and underserved—the solution offered often does not do an efficient job at meeting this challenge, leaving a firm with last minute patching and manual intervention.

Simply, this challenge is that most “descriptive” elements of the demand-supply network, such as yields, cycle times, capacity available, capacity consumed, allowable substitutions, BOM, and so on, have a start date and an end date (date effective), and the CPE must recognize that these elements will change over time. For our discussion we will use cycle time.

Previously, we used Figure 4-1 as a simple production flow to explain explosion and implosion. In that example, the cycle time was fixed over time. Let’s make the cycle time for Module_2 and Card_2 date effective (Figure 4-19). The cycle time to produce Module_2 is 8 days in duration from day 1 to day 10 of the planning horizon and 10 days from day 11 to day 25. This means that the production of Module_2 started on day 5 has a cycle time of 8 days and will complete on day 13 (5+8). But if the production starts on day 12, the cycle time becomes 10 days and the completion time will be day 22 (12+10). The cycle time for Card_2 is 4 days from day 1 to day 14 and then reduces to 2 days from day 15 to day 25. Therefore, the search engine in both explosion and implosion must account for these changing cycle times.

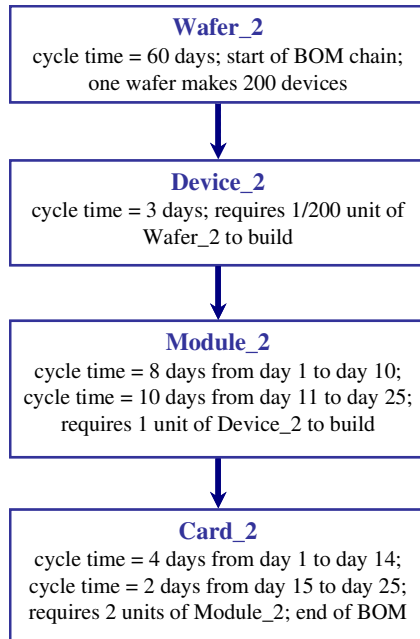


Figure 4-19: Flow for Production of Semiconductor Parts with Date Effective Cycle Times

Now let's revisit our implosion example. Manufacturing estimates four units of Device_2 will be available on day 10. If manufacturing immediately uses these four units to produce Module_2, the cycle time will be 8 days and on day 18 (10+8) four units of Module_2 will be completed. Continuing the projection process, the four units of Module_2 are immediately used to create two units of Card_2 beginning on day 18 (therefore the cycle time is 2 days) which will be available on day 20 (18+2).

What if capacity was not available on day 10 to start the production of Module_2? If capacity became available on day 11, the cycle time for Module_2 would increase from 8 days to 10 days. Therefore, a one-day delay due to a capacity restriction would result in the completion of Card_2 being delayed 3 days to day 23 (11+10+2).

Let's turn our attention to explosion. To meet demand for one unit of Card_2 on day 20, the plant must have two completed units of Module_2 available on day 18 (20 minus the cycle time for Card_2 on day 18 = 20-2 = 18). This generates an exploded demand of two units of Module_2 with a need date or due date of day 18. When does the manufacturing facility need to start the production of two units of Module_2? When we had only one cycle time for the module, we simply subtracted that value (8) from the module's need date. This would drive a Module_2 start on day 10 (18-8). Life is not that simple with date effectivity. As a starting point, we first find the cycle time for Module_2 on day 18 which is 10 days. Remember 10 is the cycle time for the production of Module_2 that starts on day 18. But we do not want to start production on day 18; rather, we want production to complete (called to stock) on day 18. So we subtract 10 from 18 and get an initial starting date of 8. Does this work? We check the cycle time for the production of Module_2 on day 8 and find it is 8 days, meaning that if we start the production on day 8 the module will be completed on day 16, two days earlier than the time we need them. We might just stop here, since we have found a feasible solution. However, another demand may need a start on day 16. We extend our search to determine if we can start later than day 8. Logically we might try day 10. The cycle time for the production of Module_2 on day 10 is still 8 days, so the modules will be completed on day 18 (10+8). Just in time!

This is not the only complexity associated with date effectivity. Two other common ones are aggregating cycle times and converting to time buckets. In cycle times, we might lump the production of Module_2 and Card_2 into one production activity (call it Mod_Crd_2). Before introducing date effectivity this was simple—just add the two cycle times and it becomes the cycle time for the new activity (8+4=12). Now we need to adjust for cycle time changes. In this case, the cycle time for Mod_Crd_2 is 12 days from day 1 to day 6, 10 days from day 7 to day 10, and 12 days again from day 11 to day 25. In time buckets, we need to convert the cycle time from daily information into number of buckets. For example, if the time bucket was 3 days in duration, we have 8+ buckets (25/3) and we need to restate each cycle time from days to buckets. Some of the buckets are located on a date effective boundary.

4.2.10 Other Technical Challenges

Other challenges include, but are not limited to, demand perishability, squaring sets, soft capacity constraints, alternative capacity, pre-emptive versus weighted priorities, splitting demand to match partial delays in supply, stability, express lots, delay assembly to test, dispatch lots, foundry contracts (Figure 4-20: service model; multiple exit demands along the same bill of material supply chain), risk based inventory policy, multi-supplier sourcing using inventory, WIP projection, rules governing purchase order change recommendations (allow for date change, quantity change, both or neither).

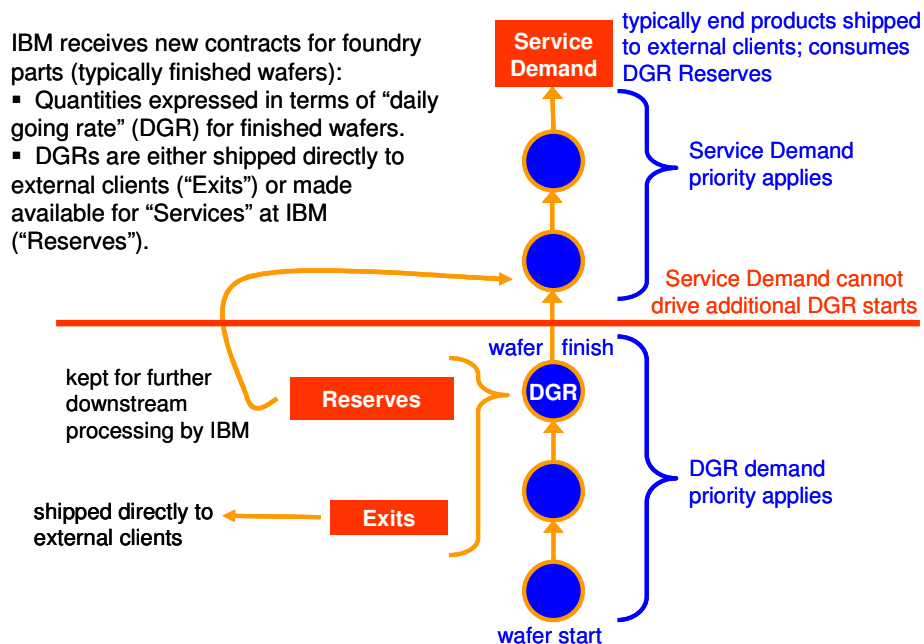


Figure 4-20: Service and DGR Demands for Foundry Contracts

4.2.10 Challenge of Uncertainty – the Stochastic Nature of the Demand-Supply Network

Uncertainty (in parameters, estimated supplies, projected demands, etc.) is no doubt another critical challenge, but we will only briefly touch on this topic in this section.

In the near term, uncertainty may force planning applications to take a conservative approach to risk. For example, if (a) the average binning percentage for fast parts is 30%, (b) it has

a uniform variability of plus/minus 10%, and (c) there are 100 parts currently being tested, then the actual number of fast parts that we will get from this specific manufacturing activity ranges from 20 to 40. Although on average we will get 30 fast parts from the sorting operation, 50% of the time the actual number of fast parts will be less than 30. Working with the average value is fine over a moderate or long time frame, but it is impractical if your time frame is only one day. One proposed method called cycle variation looks at taking a more conservative approach with high priority demands than with low priority ones. The downside of this method is that “lower priority demands” had a higher probability of the central planning engine determining they could be met on time—a small problem!

For long range planning, executives would prefer to understand the range of possible outcomes and their likelihood, instead of being given a point estimate. There is a range of work going on (for example, Kempf 2004) with some involving stochastic optimization while others pulling from dispatch scheduling of wafer fabricators (Burda et al 2007) and wafer fabrication capacity planning (Zisgen 2005) and still others exploring inventory policy (IBM 2005).

4.2.11 The Big Bang Approach and General Challenge of Reduced Solution Times

We refer to today’s supply chain management central planning process as the “big bang” approach. An enterprise creates a centralized process and data representation of the firm at some point in time. Then, through some combination of automated models and manual processes, the enterprise creates a “global plan.” Some firms execute a global planning process a few times each day; others do it once a month and take a week to create the plan. In both cases, the new plan arrives after a reasonable amount of effort and replaces the old plan in its entirety. A new “universe” is created via the “big bang!”

This approach creates the general challenge of reducing solution times with parallel processing and faster algorithms and machines, improving accuracy with smaller time buckets or units of granularity, with more robust representations of the demand-supply network, or with latest techniques such as constraint programming, handling the stochastic nature of information, and automating some aspects of plan review.

4.2.12 Beyond the Big Bang—Supply Chain Physics of the 20th Century Meets Its Quantum Revolution

At the start of the 20th century, physicists learned that we do not live in a clockwork universe [84]. The same can be said for supply chain modelers at the start of the 21st century. Although centralized processes and models have increased organizational effectiveness, there are clear limits and we are rapidly reaching them.

Effective centralization refers to the ability to take into consideration all aspects of the decision situation simultaneously and generate an optimal or at least a very good solution. To be effective, a central solution requires a synchronized current view of the entire decision landscape, the ability to handle complex tradeoffs, and a reasonably fast runtime performance. Gaps exist and are often created by time lags, summarization, performance, triggers, and formulation. By triggers, we refer to the event that “wakes up” at a specific time during the day and runs the central solution. Once a week, once a day, or once every 3 days, the central solver executes, but the decision to execute is made without any knowledge or monitoring of events since the last execution. Formulation gap refers to the inability to formulate key decision questions in such a manner that lends itself to a central solution as opposed to a sequence of negotiations or collaborations. The reality is, even if we could get the big bang models to finish execution in 5 minutes, an organization would run it once a day at best to match demand with supply and synchronize the

enterprise. As the time between runs decreases (for example, from once a week to once a day), the following limitations become apparent:

1. Understanding and repairing the plan:
 - 1.1. Why does the plan give these results?
 - 1.2. What demands are not being met (alerts) and why?
 - 1.3. What actions can I take to improve the plan?
 - 1.3.1. Identify and book actions to improve the supply posture as it relates to demand.
 - 1.3.2. Less significant items such as inventory picture.
 - 1.4. Monitoring the repair actions as they are being executed.
2. The nature (quality) of the plan – no plan to plan continuity:
 - 2.1. No checks, filters, or alerts (CFAs) on demand information / signals.
 - 2.2. No CFAs on supply information / signals.
 - 2.3. No CFAs on changes in production specification or business policy.
 - 2.4. Each plan is built from scratch.
 - 2.5. No build-in dialogue with other key providers of input, such as
 - 2.5.1. Projected supply.
 - 2.5.2. Projected demand.
 - 2.5.3. Capacity estimation.
3. Observe both 1 & 2 require an incremental matching (planning) or net change engine.

The following example shows how a “simple” repair action can quickly become complex.

In **Table 4-3** we have two demands for the same part (P111): D001 with a due date of 4/4/2006, priority of 2, and quantity of 80 and D002 with a due date of 4/5/2006, priority of 1, and quantity of 100.

Demand for Part P111			
ID	Date	priority	quantity
D001	04/04/06	2	80
D002	04/05/06	1	100

Table 4-3 Demands for Part P111

In **Table 4-4** we have two anticipated supplies for part P111: S00A and S00B with an anticipated delivery date of 4/4/2006 and 4/6/2006; the quantity is 100 units each.

Supply for Part P111			
ID	Date	priority	quantity
S00A	04/04/06	na	100
S00B	04/06/06	na	100

Table 4-4 Supplies for Part P111

Initially, we assign S00A to D002 (**Figure 4-21**) since D002 has a higher priority and the other supply S00B would arrive 1 day after the due date for D002. This assignment resulted in D001 being met 2 days late.

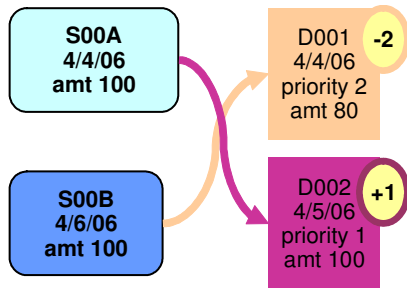


Figure 4-21: Initial Assignment of Supply to Demand for Part P111

Figure 4-22 demonstrates a simple repair action to meet demand D001 on time: the analyst requests that the S00B supply be completed two days earlier on 4/4/2006.

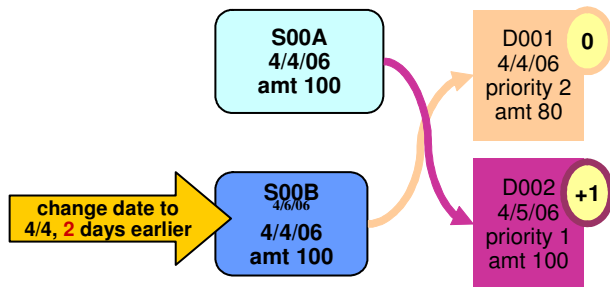


Figure 4-22: Simple Repair to Meet Demand D001 on Time

Figure 4-23 shows a smarter repair action. First, the analyst switches the assignment of supply to demand – supply S00A is assigned to D001 and supply S00B is assigned to D002. Then the analyst requests that supply S00B be expedited only 1 day to 4/5/2006 to meet D002 on time.

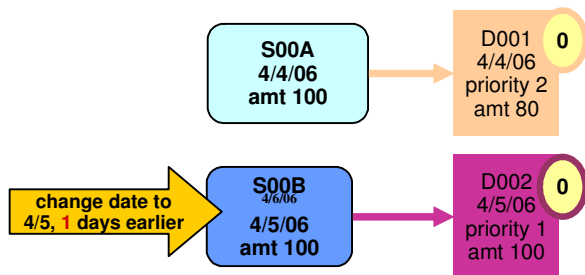


Figure 4-23: Smarter Repair Action to Meet Both Demands On Time

To take organizations to the next leaps in efficiency requires a substantive adjustment in approach. Just as 19th century physics had to adjust its equations and formulations for special relativity, general relativity, and quantization, supply chain solvers must learn to accommodate techniques from intelligent agents and sense and respond and learn to incorporate the concept of collaboration into their solution processes. Collaboration refers to an iterative process that focuses on finding a satisfactory solution. The next search step depends on prior steps and may involve back tracking. Often, the step involves negotiating a temporary change in a subset of the rules governing the game for a limited period of time, and typically contingency occurs to handle uncertainty.

5 Factory Planning Basics and Challenges

5.1 Overview of Factory Planning

Factory planning, like central planning, is focused on matching assets with demand to determine what commitments can be made to the business and what opportunities to chase. There are typically three major types of planning: aggregate planning; deployment or near term tool planning, and WIP Projection.

5.1.1 Aggregate Factory Planning

Aggregate planning [1, 79, 87, 88] is focused on assessing the ability of the factory to meet certain demand looking to identify “broken” toolsets and creating the capacity inputs required by central planning model. Demand is typically stated as a starts profile and a lead (cycle) time commit for each part. For example, the factory has been asked to start 100 of Part A each day with a lead time of 20 days and start 50 of Part B with a lead time of 10 days. Historically, the focus is typically to identify “broken” tool sets (groups of tools currently without sufficient capacity to meet this load (starts) and cycle time). Once these tools sets are identified there are two options: actions are taken to add capacity or central planning modifies the load profile. Typically (but not always) this is a slow and reasonably static process compared to the more dynamic central planning process. It operates more like an MRP (which simply tells the central planner what needs to be done to achieve a give demand), then a “best can do” model (which searches a wide set of options to best meet prioritized demand without violating constraints). That said, there is strong interest and need to upgrade this process to dynamically handle such factors as demand priority, cycle time versus load trade-off, and partially shared operations between tools. The basic steps are

8. Capturing representative product routes- sequence of operations, raw process time at the operation, and tool set consumed for each key operation.
9. Capturing a specific factory load – typically given in starts and lead time or cycle time commit.
10. Tool set characteristics: number of tools; tool availability; operating curve; deployment; reentrant flow
11. Allocation of tool sets to parts or families either as user input or based on history
12. A model that captures key relationships
13. Executing the model to determine how this load impacts toolsets
14. Providing a capacity statement (resource consumption rates and resource available) to central planning

Why is this different then central planning? Aggregate factory planning deals with

- the actual route (sequence of manufacturing actions to produce a finished part)
- the operating curve which establishes the trade-off between cycle time and tool utilization / capacity available.
- Partially Shared Operations (PSO) or Cascade which refers to different toolsets that overlap in the manufacturing operations they service, but the actions are not identical.
- Deployment Variability
- Product mix transitions
- reentrant flow
- determining a method to summarize capacity (required and available) to fit the traditional simpler approach to handling capacity in central planning

All but the first item is described in more detail in later sections and make up much of the core of the ‘ongoing challenges’.

5.1.2 Deployment or Near Term Tool Planning

Deployment refers to determining which operations each tool will be qualified to handle over the short term. Typically a tool can service many operations, but a factory will limit the number of operations it is “allowed” to service to

- focus tools on the operations they are best suited to handle (speed, yield, etc)
- ensure a tool remains qualified to handle an operation by regularly servicing this operation
- tool specific restrictions which may limit a tool from serving some operations based on the operations it can currently service
- reduce work load on manufacturing engineering to maintain tool qualification
- make dispatching simpler by reducing options

5.1.3 WIP Projection

All central planning engines require each factory to project when current WIP in the line will reach either completion or staging point – typically called projected finish. The typical process involves capturing the remaining steps (route) until the lot reaches its end point and then making an estimate of how long the lot will need to finish these steps.

5.2 Challenges and Opportunities for Aggregate Factory Planning

Over the past twenty years organizations have put significant energy into making smarter decisions in their enterprise wide central planning and “available to promise” process to improve responsiveness (more effective use of assets and more intelligent responses to customer needs and emerging opportunities). However, firms have put limited energy into factory floor decisions and capacity planning, and almost none into a tighter coupling between factory and central planning. Much of the recent work to improve factory performance has attempted to implement core lean planning [10] concepts of (a) elimination of variability, (b) establishing uniform flow (every part every interval), (c) supermarket goods flow (kanbans), and (d) elimination of due dates. Clearly every factory will run “better” with steady output and predictable lead times—however the real world always injects variability that sets the price of implementing such methods as reduced responsiveness and/or excess capacity.

The net result is that many factories still operate with the mind set: “establish a set of starts for the month; set a fixed schedule with target outs; measure actual outs versus target outs”. However for this approach to work, demand must be accurately forecasted over an extended period of time and uniformly spread across time; all lots travel at the same speed, tool sets operate with clockwork precision (never suffering “surprises”); and the flow of parts in the line (even with stable capacity) never creates “piles” or “gaps” due to the variations (for example batch versus single lot tools) intrinsic in the manufacturing process. In today’s world, accurate detailed forecasts of demand remain an illusion, even the best factories have “tool set surprises” (breakdowns and excursions), product mix introduces variability in speeds, and the competitive nature of the market precludes carrying excess capacity and insists on responsiveness. Those demand supply networks that can get their factories more engaged in responsiveness while recognizing the importance of “tools” and “output” will flourish; they will eliminate the variability that matters—a failure to deliver a part on its committed date and the inability to capture a market opportunity that could be handled with “intelligent” factory decisions.

5.2.1 Some Inconvenient Complications

In the traditional world of central planning, capacity is modeled with

- simple linear equations where the two primary inputs are resource required and resource available (for every desk produced we need 4 legs, 1 top, and varnish)
- the cycle time or lead time is fixed and disconnected from capacity
- capacity allocation is usually focused on manufacturing releases, not work in progress

Factory features such as the operating curve, reentrant flow, partial sharing of operations between tools, and deployment decision are a level of complexity that bound the effectiveness of linear models.

5.2.1.1 Recognizing the trade-off between capacity and cycle time – headaches

5.2.1.1.1 Review of the Operating Curve

When variability exists either in arrivals or services there is a trade-off between server (tools, people) utilization and the lead or cycle time to complete an activity or service → the higher the utilization the longer the cycle time. Since higher planned utilization translates to more effective output the trade-off can be reframed as effective capacity available versus cycle time. The curve that describes this trade-off is called the **Operating Curve (OC)**. Cycle time is often measured as a cycle time multiplier (CTM), where CTM equals total elapsed cycle time divided by raw process time (RPT). Typically the curve is almost flat for low utilization levels, then spikes sharply upward from the steep part of the curve. There a number of equations that can generate this curve, but the one we will use is

$$CTM = 1 + offset + \alpha \left[\frac{util^M}{1 - util^M} \right] \quad (\text{Morrison and Martin [65]})$$

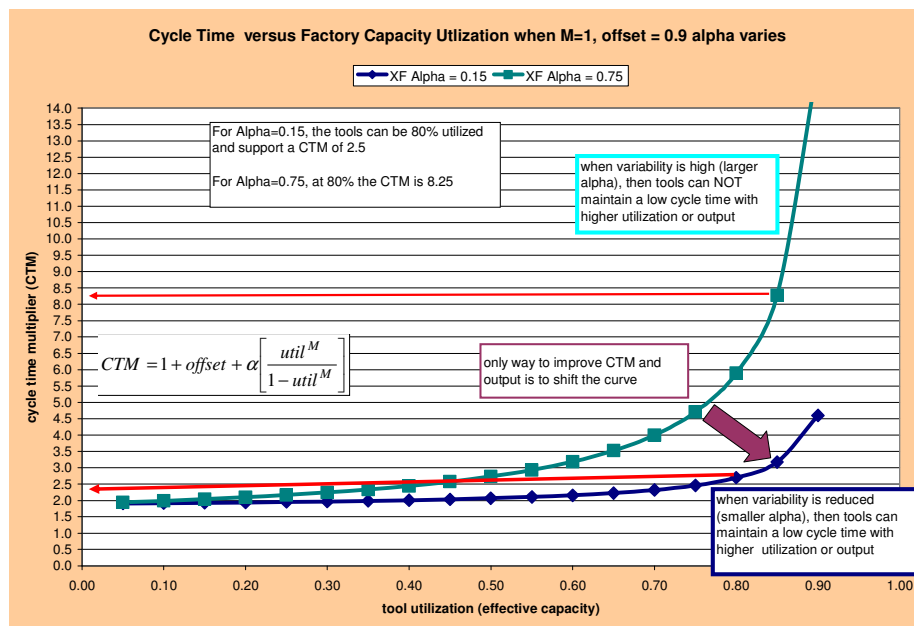


Figure 5.1: Cycle Time versus Factory Capacity

- **CTM** is the cycle time multiplier of RPT – measure of cycle time
- **util** is the fraction of tool utilization in the entity – facility, tool set, checkout clerks, etc
- **offset** represents several of aspects of the process that generate wait time and cannot be eliminated. For example: travel time, hold hours, post processing hours relative to total RPT. A typical value for offset is 0.9. When offset is 0.9 this sets the minimum CTM at 1.9
- **M** is the number of identical parallel machines or servers. Typically this value ranges from 1 to 4. Fournier, Morrison, and Martin report values greater than 3 or 4 (even when the number of tools exceeds 4) work best.
- **α** represents the amount of variation in the system and controls how long the curve stays flat. The lower the value of α the less variation and longer the curve stays flat. Common values for α range between 0.35 and 0.65.

Solving for Util, we have
$$Util = \left(\frac{CTM - (offset + 1)}{CTM - (offset + 1) + \alpha} \right)^{\frac{1}{m}}$$

5.2.1.1.2 Linking Capacity and Cycle Time

Assume that the product XYZ is processed 5 times by tool set AAA during its production route. Each time a widget goes through tool set AAA it is referred to as a “pass”. In this case product XYZ has 5 passes on tool set AAA. Additionally, assume 100% process yields and that the average RPT for each XYZ widget on tool set AAA is 2 units. In steady state, this makes for a total RPT required per day of 10 (=5x2) units per widget of XYZ on tool set AAA. Assume we start 1 widget per day and we have 10 units of capacity, what would the cycle time be? The CTM equation makes it clear the cycle time would be infinite since the capacity required matches capacity available making tool utilization 100%.

The business states it wants to run this product with a CTM of 4.0. This requires some portion of time that the tool set is available to produce, but does not have WIP. How do we incorporate that into the planning process?

We calculate a burden or uplift factor (ULF) per widget based on the target CTM and the specific characteristics of the Operating Curve for this tool set. Assume the Operating Curve for this tool set has offset=1, alpha=0.5, and m=1. Using the above equation to solve for UTIL, the required utilization to achieve the CTM target of 4.0 is 0.80 (80%). For each unit of raw capacity required, we need 1.25 units available to meet the CTM target. The value 1.25 is the uplift factor (ULF) and determined by:

$$ULF = 1 / \text{tool\_utilization\_meet\_cycle\_time\_target\_from\_opcurve}$$

If we have 250 units of capacity available per day, how many widgets can we start per day at committed cycle time? The answer is $20 = \{250 / (1.25 \times 10)\}$.

If the business wanted to achieve a cycle time of 3.5, how many widgets could it start per day? Using the same equation for UTIL, the utilization required to achieve a cycle time of 3.5 is 0.75 (75%). The ULF is $1.333 = 1 / 0.75$. The maximum number widgets per day at this cycle time commit is $\{250 / (1.333 \times 10)\} = 18.75$. Shorter cycle time equates to reduced widget starts.

Alternatively, instead of decreasing the capacity available, we can uplift the capacity required per unit of production. For the 4.0 cycle time commit, we uplift 10 units per day per widget to 12.5 (10×1.25). For the 3.5 commit, the 10 units is uplifted to 13.33 (10×1.33). Figure 4 gives us the uplifted capacity required to input to the central planning process based on the cycle time the business wishes to achieve.

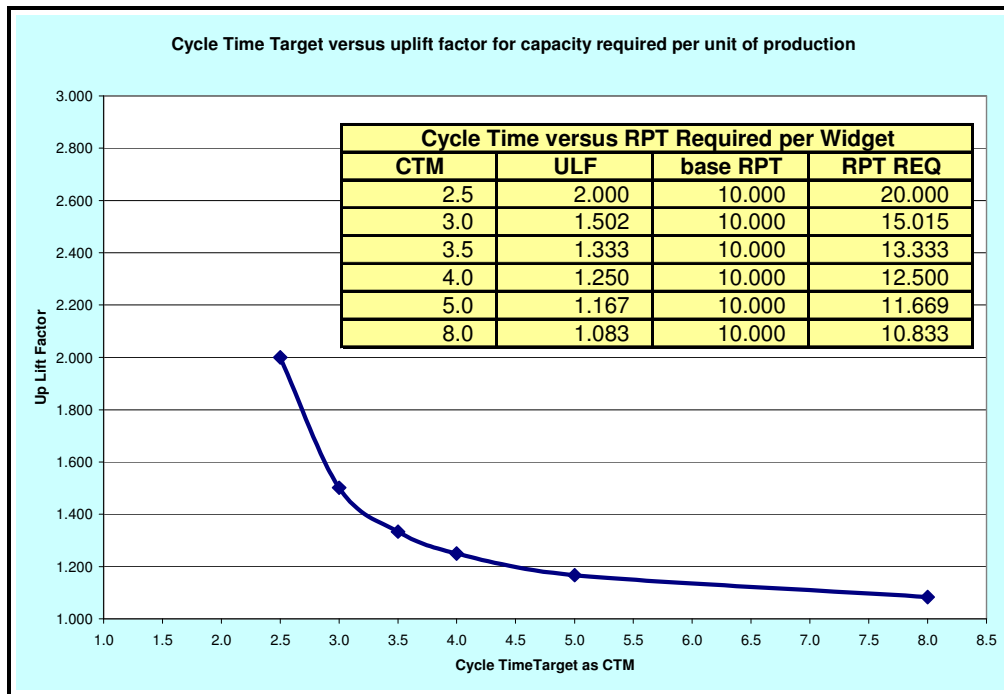


Figure 5.2: Uplift in Capacity Required Based on Cycle Time Commit

This example makes it clear that the assumption in typical central planning processes that cycle time and capacity are independent is not correct – the two are clearly coupled. We can view this as classical planning meets its uncertainty principle. It is a rich ground for improved responsiveness and headache for classical planners. Since lean advocates believe that all variation can be eradicated, they have no awareness of an operating curve, and no methods to capture this opportunity. It is like attempting to ignore special and general relativity and still produce GPS locations (Pogge 2009).

5.2.1.2 Reentrant Flow

From a PSD perspective the critical insight is the repetitive use of the same core processes across the front end and back end and hence the same core equipment sets, is called “reentrant flow.” This generates the complexity of partially shared operations (PSO or cascading) and deployment decisions. The following example will illustrate these issues. **Figure 5-3** has a simplified set of major manufacturing activities to build one circuit layer.

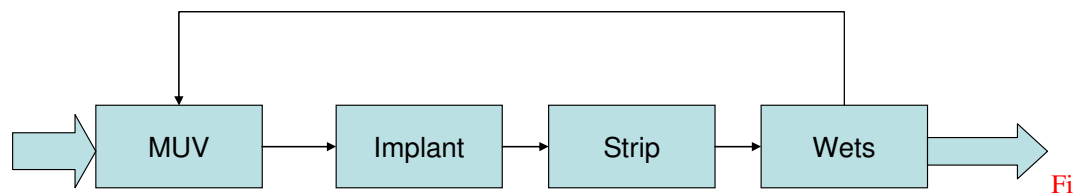


Figure 5-3 typical steps in building a circuit layer

This core set of activities is repeated multiple times to build a part and the number of iterations is different between parts. **Figure 5-4** shows the number of iterations through these same steps for Product A (three passes) and Product B (two passes)

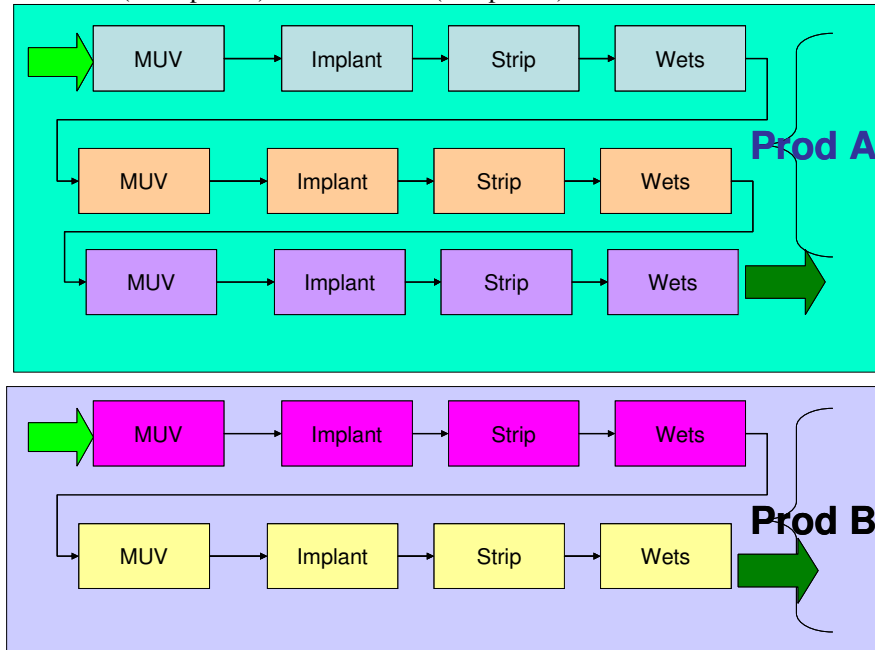


Figure 5-4: Example of reentrant flow for typical circuit construction

In this example Product A requires three passes through the MUV toolset and Product B two passes. Assume MUV is serviced by three tools (A, B, and C). Typically not all tools handle all operations (passes). **Figure 5-5** demonstrates the concept of partially shared tools sets for related, but different manufacturing operations. In this example there three tools (A, B, and C) to handle the MUV step. But not all tools can handle all steps. This is the deployment decision.

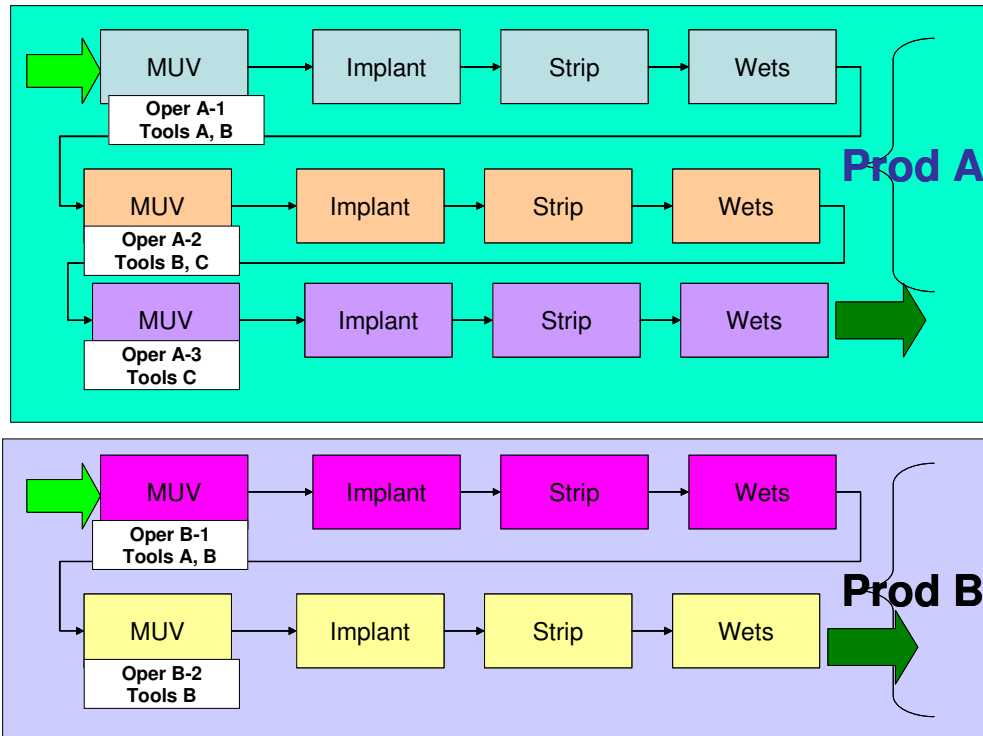


Figure 5-5a Example of reentrant flow with Partial Shared Operations (PSO) or Cascading

In table form, the deployment decision would be represented as follows

Deployment Information for Circuit Layers				
	Tool A	Tool B	Tool C	no tools covering oper
oper A-1	1	1	0	2
Oper A-2	0	1	1	2
Oper A-3	0	0	1	1
Oper B-1	1	1	0	2
Oper B-2	0	1	0	1
number ops tool covers	2	4	2	

Figure 5b: Partial Shared Operations in Table Form

The decision about which tools can handle which operations (deployment decision) is essentially determining which cells in table 3.1 get a 1 and which get a 0. Typically a tool can service many operations, but a factory will limit the number of operations it is “allowed” to service which is called the deployment decision.

5.2.1.3 Partially Shared Operations (PSO) – Cascading

In Table 5.1 we have three tools (A, B, and C) to service seven operations (001 – 007) creating a 7 by 3 grid. A 1 indicates that this tool can service this operation and 0 it can not. Assume there are no other tools that can service these operations and there are no other operations that these tools can service. This is called PSO or Cascade group since all of there are no tool/operation subsets that can be found that are independent – at some level each of these operations and tools are connected. From a tool perspective

- Tool A and Tool B “share” operations 1 and 2
- Tool A and C do not share any operations
- Tool B and Tool C share operations 3 and 5
- Therefore the connection between Tool A and Tool B is a second order link.

Table 5.1: Deployment Information for PSO Group				
	Tool A	Tool B	Tool C	no tools covering oper
oper001	1	1	0	2
oper002	1	1	0	2
oper003	0	1	1	2
oper004	0	0	1	1
oper005	0	1	1	2
oper006	1	0	0	1
oper007	1	0	0	1
number ops tool covers	4	4	3	

Table 5.2 has the same tools and operations as table 2.1, but with a different deployment allocation. In this situation there is overlap in operations between Tool B and Tool C, therefore there are two PSO groups. Tool A and Tool B (and the operations they service – 1, 2, 5, 6, and 7) and Tool C (operations 3 and 4).

Table 5.2: Deployment Information for 2 PSO Groups				
	Tool A	Tool B	Tool C	no tools covering oper
oper001	1	1	0	2
oper002	1	1	0	2
oper003	0	0	1	1
oper004	0	0	1	1
oper005	0	1	0	1
oper006	1	0	0	1
oper007	1	0	0	1
number ops tool covers	4	3	2	

An additional wrinkle to this feature is the raw process times for the same operation may differ

between tools.

Table 5.3: RPT information			
	Tool A	Tool B	Tool C
oper001	4	20	5
oper002	15	20	6
oper003	10	15	8
oper004	10	9	20
oper005	5	5	5
oper006	8	10	10
oper007	10	10	10

5.2.1.3.1 Do I have enough capacity to handle average workload?

The critical aggregate planning question is do I have enough capacity to handle average workload? How do I mix and match the workload at each operation to the tools available to meet the most demand?

Table 5.4 has the average daily workload for operation and the average capacity available for each tool.

Table 5.4: Average Daily Demand and Capacity Available				
	Tool A	Tool B	Tool C	average daily workload
oper001	4	20	999999	40
oper002	15	20	999999	40
oper003	999999	15	8	40
oper004	999999	999999	20	40
oper005	999999	5	5	50
oper006	8	999999	999999	50
oper007	10	999999	999999	50
capac avl	720	1152	1296	

The key challenges for the factory planner are:

- Determine if the workload can be allocated across the tools in such a way that all of the workload can be allocate without violating capacity constraints
- If insufficient capacity exists
 - find the optimal mix of workload that can be met without violating capacity constraints
 - find the optimal allocation that either minimizes additional capacity needed incorporating some type of fair share of pain

Table 5.5 has one feasible solution. The capacity limits are not violated, but 50 units of workload can not be serviced.

Table 5.5: Allocation Decision and Results where capacity limits are not violated							
	Tool A	Tool B	Tool C	demand met	constraint	average daily workload	delta
oper001	23.0	17.0	0.0	40.0	≤	40	0.0
oper002	0.0	30.0	0.0	30.0	≤	40	10.0
oper003	0.0	10.0	30.0	40.0	≤	40	0.0
oper004	0.0	0.0	40.0	40.0	≤	40	0.0
oper005	0.0	10.0	30.0	40.0	≤	50	10.0
oper006	40.0	0.0	0.0	40.0	≤	50	10.0
oper007	30.0	0.0	0.0	30.0	≤	50	20.0
cap used	712.0	1140.0	1191.0		total unmet dmd		50.0
constraint	≤	≤	≤	assign lots to tools to minimize unmet demand without exceeding capac avail or workload			
cap avl	720.0	1152.0	1296.0				
delta	8.0	12.0	105.0				

Table 5.6 has one solution where all workload is met, but capacity limits are exceeded.

Table 5.6: Allocation Decision and Results where capacity limits are not violated							
	Tool A	Tool B	Tool C	demand met	constraint	average daily workload	delta
oper001	23.0	17.0	0.0	40.0	≤	40	0.0
oper002	0.0	40.0	0.0	40.0	≤	40	0.0
oper003	0.0	10.0	30.0	40.0	≤	40	0.0
oper004	0.0	0.0	40.0	40.0	≤	40	0.0
oper005	0.0	10.0	40.0	50.0	≤	50	0.0
oper006	50.0	0.0	0.0	50.0	≤	50	0.0
oper007	50.0	0.0	0.0	50.0	≤	50	0.0
cap used	991.8	1340.0	1241.0		total unmet dmd		0.0
constraint	≤	≤	≤	assign lots to tools to minimize unmet demand without exceeding capac avail or workload			
cap avl	720.0	1152.0	1296.0				
delta	-271.8	-188.0	55.0				

5.2.1.4 Deployment Decision

This decision is essentially determining which cells in table 2.1 get a 1 and which get a 0. Deployment) refers to determining which operations each tool will be qualified to handle over the short term. Typically a tool can service many operations, but a factory will limit the number of operations it is “allowed” to service to

- focus tools on the operations they are best suited to handle (speed, yield, etc)
- ensure a tool remains qualified to handle an operation by regularly servicing this operation
- tool specific restrictions which may limit a tool from serving some operations based on the operations it can currently service
- reduce work load on manufacturing engineering to maintain tool qualification
- make dispatching simpler by reducing options

5.2.1.4.1 The decision is how to best allocate tools to near term work load

In this example we have three tools (A, B, and C) and seven operations (oper001 – oper007) to handle the current WIP at this tool set. Table 5.7 shows the current deployment decisions made by the factory planner. A 1 means this tool (column) is able to service this operation (row). For now these decisions are fixed.

Table 5.7: Deployment Decision				
	Tool A	Tool B	Tool C	# tools covering oper
oper001	1	1	1	3
oper002	1	1	0	2
oper003	0	1	0	1
oper004	0	0	1	1
oper005	1	1	1	3
oper006	1	0	0	1
oper007	1	0	0	1
number ops tool covers	5	4	3	

Table 5.8 provides all of the key pieces of information for the model. The value in oper / tool cell is the raw process time (RPT) that the tool requires to process one lot at this operation. For example, the RPT for Tool A to process one lot at oper001 is 4 time units. A value of 9999999 indicates this tool / operation combination is not currently active, corresponding to a 0 in table 15. The last column (# lots) provides the number lots requiring service at that operation at this point in time. For example there are 40 lots at oper001 waiting to get on either Tool A, B, or C. The last row ("capac avl") provides the number of time units of capacity available for that tool over the service period. Tool A has 720 units available.

Table 5.8: RPT, Lots, Available Capacity				
	Tool A	Tool B	Tool C	# lots in WIP waiting to be processed
oper001	4	20	5	40
oper002	15	20	9999999	30
oper003	9999999	15	9999999	10
oper004	9999999	9999999	20	60
oper005	5	5	5	10
oper006	8	9999999	9999999	200
oper007	10	9999999	9999999	200
capac available at tool	720	1152	1296	

Table 5.9 has the basic "what if" model. The value in each tool / operation cell is the *business decision*- the number of lots the tool is assigned to handle for this operation over some time period. For example, Tool B will handle 30 of the lots that require service at oper001; 10 lots for oper003, and 0 lots for oper005. These 21 values (cells) (7 operations by 3 tools) represent an allocation decision.

Table 5.9: Allocation Decision and Results							
	Tool A	Tool B	Tool C	lots served	goal type	# lots goal target	delta from target
oper001	0.0	30.0	10.0	40.0	=	40	0.0
oper002	3.0	0.0	0.0	3.0	=	30	-27.0
oper003	0.0	10.0	0.0	10.0	=	10	0.0
oper004	0.0	0.0	60.0	60.0	=	60	0.0
oper005	0.0	10.0	0.0	10.0	=	10	0.0
oper006	90.0	0.0	0.0	90.0	=	200	-110.0
oper007	0.0	0.0	0.0	0.0	=	200	-200.0
cap used	765.0	800.0	1250.0		total unmet dmd		-337.0
goal type	≤	≤	≤				
cap maximum	720.0	1152.0	1296.0				
delta cap goal	-45.0	352.0	46.0				

The results of these decisions are found in column 5 (lots served) and row 10 (cap used). The “lots served” column is the total number of lots served for this operation across all tools. For example 40 lots at oper001 will be served – 0 on Tool A, 30 on Tool B, and 10 on Tool C (40=0+30+10). The row “cap used” provides how much capacity was used for each tool. This is the sum of the product of the allocation times RPT (table 16). For tool A, the cap used is 765 = (0*4) + (3*15) + (0*9999999) + (0*9999999) + (0*5) + (90*8) + (0*10).

The last component of the model is comparing the results of the business (allocation) decision made with the goals of the factory. The factory has two goals: service as many lots as possible and use all available capacity.

The “lots served” goal comparison information is in columns 6 (goal type), 7 (target), and 8 (delta). Our goal is to service all lots waiting (=). The target is the number of lots that are currently waiting (last column in table 16). The result is posted in the “delta” column which is simply “lots served” minus “target”. For example, for oper002 the value is -27, since our target was 30 and the actual number of lots served was 3 (-27=3-30). This tells us the current allocation decision leaves 27 lots waiting at oper002 “still waiting”. The last value in the delta column is the total unmet demand based on the current allocation decision. The value -337 is simply the column sum.

The “capacity goal” information is in the last 3 rows (goal type, cap maximum, and delta cap goal). It is a constraint – do not make allocation decisions that exceed available capacity. The target is the capacity available for each tool (last row in table 16). The result is posted in the last “delta” row which is simply “cap used” minus “cap maximum”. For example, for Tool B the value is 352, since the actual cap used was 800 and the maximum capacity was 1152 (352=1152-800). This tells us the current allocation decision leaves 352 units of capacity at Tool B idle.

This simple model enables planners to manually assess the quality of their current deployment situations and establish guidelines for manufacturing on how to allocate lots amount tools by trying different deployment decisions (table 15) and or different allocation decisions (table 17). Typically a planner will try different allocation decisions (leaving the deployment decision unchanged) to determine how to best allocate WIP to tools to meet prioritized demand and then send guidelines to manufacturing. Table 18 shows an improved allocation (red is changed allocation decisions) plan eliminating overusing capacity on tool A and reducing overall unmet demand from 337 to 330.

Table 5.10: Revised Allocation Decision and Results							
	Tool A	Tool B	Tool C	lots served	goal type	# lots goal target	delta from target
oper001	0.0	37.0	3.0	40.0	=	40	0.0
oper002	0.0	10.0	0.0	10.0	=	30	-20.0
oper003	0.0	10.0	0.0	10.0	=	10	0.0
oper004	0.0	0.0	60.0	60.0	=	60	0.0
oper005	0.0	10.0	0.0	10.0	=	10	0.0
oper006	90.0	0.0	0.0	90.0	=	200	-110.0
oper007	0.0	0.0	0.0	0.0	=	200	-200.0
cap used	720.0	1140.0	1215.0		total unmet dmd		-330.0
goal type	≤	≤	≤				
cap maximum	720.0	1152.0	1296.0				
delta cap goal	0.0	12.0	81.0				

Additional complications include different lot priorities, multiple periods, and partial deployments (which occur during a phase in).

To enable the planner to model the impact of deployment decisions, we need to couple the decisions made in table 5.7 and table 5.9. He or she can change the deployment decision (table 5.7); then revisit the allocation decision (Table 5.9); then assess the impact on the WIP waiting to be serviced.

The Planned Deployment Decision is Worthless without Execution: In this example we have made a decision to run all of the lots for oper006 on Tool A. We have made this decision because lots at oper003 can only run on Tool B and lots at oper004 can only run on Tool C. However, on the factory floor doing this type of analysis would at best be difficult. Second, from the floor's point of view, the RPT for lots at oper006 is the same (5) for all three tools. Therefore a casual analysis would make the floor indifferent to which tool handled lots at oper006 --- with dire consequences to the factory.

6. Basics and Challenge of Dispatch (assigning lots to tools)

6.1 Basic Architecture

The work by Sullivan, Kempf, Fowler, Graves, Wein, Glassey, Bitran, and many others in the 1980s and early 1990s in planning, scheduling, and dispatching wafer fabricators has been extensively written about the basics of wafer fabrication flow (reentrant); the core considerations in assigning a lot to a tool; and the basic architecture of a rule based dispatch system is now well known. The core architecture of a dispatch system and the method of obtaining and organizing knowledge has remained remarkably similar to the original work done by Sullivan [77]. Figure 6.1 provides a basic overview.

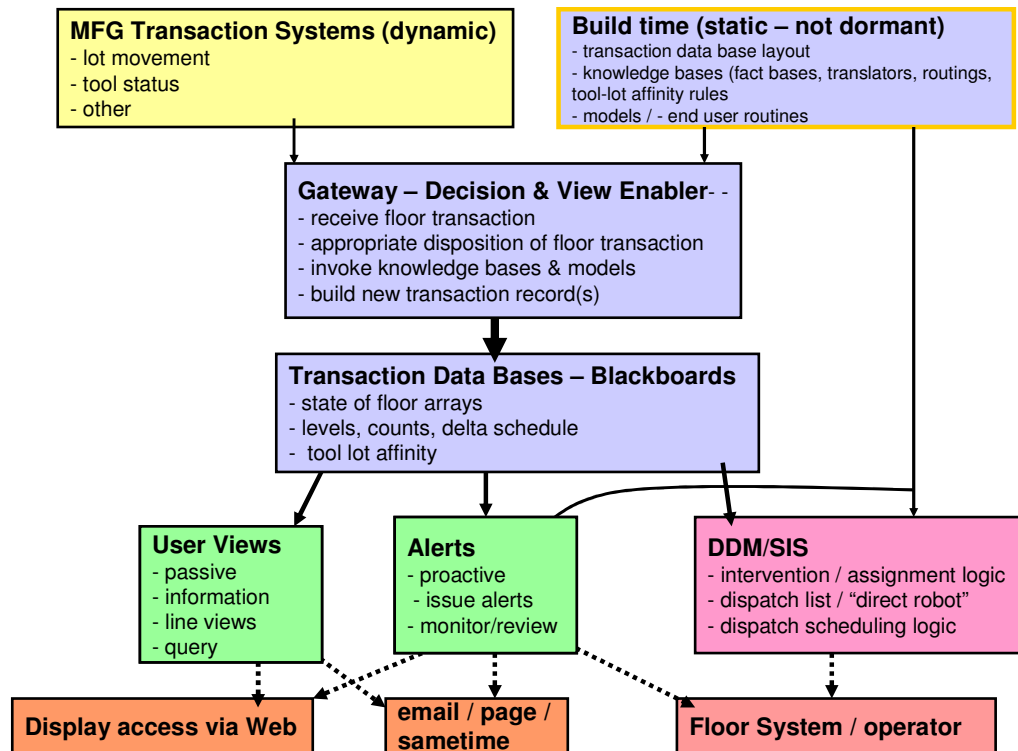


Figure 6.2: Core flow for rule based dispatch

The problem has fundamentally remained true to John Fowler’s 1992 (NSF Workshop October 1992 New Hampshire) description: “Dynamic Production Planning and Scheduling is needed in semiconductor manufacturing because of the complexity of the manufacturing processes including factors such as unreliable equipment, batching, reentrant flows, rework, yield loss, hot lots, combination of production, engineering and R&D lots, and varying product mix and start rates. In addition to these factors, future wafer fabs will probably be more automated than current factories. This will require computers to control the flow of material through a fab instead of humans. While all of the factors mentioned above make the real-time control of a factory necessary, they also make it very complicated to determine what the best decisions are. There are, however, a few things that we have going for us.

1. Except for rework, most of the flow in a FAB is deterministic (instead of probabilistic as in a job shop).
2. The processing time per wafer or per lot or per batch is very nearly deterministic, so that once processing begins; we can get a very good prediction of when the processing will end.
3. The shop floor control systems in place in current wafer fabs provide much of the information we need in order to make good decisions

6.2 Fundamentals of the Assignment Decision

Historically, the easiest way to conceive of dispatch is when a tool becomes free (moves from a busy state to a non busy state or comes back on line) lots compete to be the “next on the tool”. In practice, other triggers, such as the change in status of a lot or elapsed time, can trigger

the logic to review and assign lot(s) to a tool. **Figure 6.2** illustrates the basics of lots waiting for a tool at a tool set.

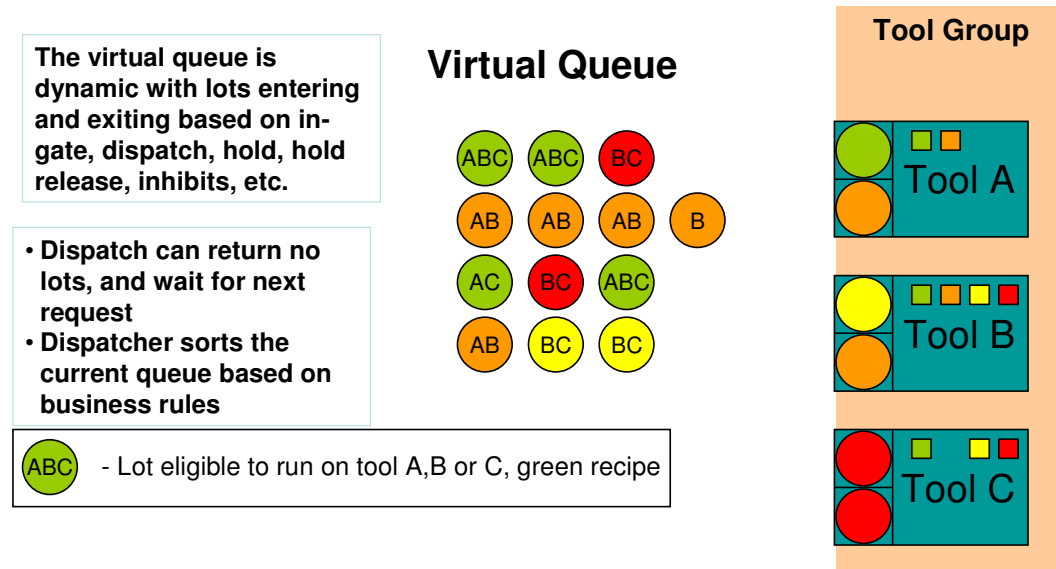


Figure 6.3: Basics of lots waiting to be dispatched

Fundamentally this decision process must first determine which lots are eligible to run on which tools and then narrow the selection process based on business guidelines (Due dates, Business Rules, dedication strategies, phase in tools); Process requirements (production paths, reticle available, tool specific inhibit); and Intrinsic properties of tools which significantly impact throughput (for example batch size and trains); and occasionally synchronization of events across the Fab. We can break the key elements down as follows:

1. tool – lot affinity
 - 1.1. what lots can run on this tool, what tools can handle this lot
 - 1.2. what are preferred tools, what are preferred lots
 - 1.3. manufacturing engineering requirements
 - 1.3.1. counts (avoid too many lots on certain tools)
 - 1.3.2. time limits (tool requires re-qualification)
 - 1.3.3. process time windows (lot must finish a sequence of steps within a time limit)
 - 1.3.4. special customer specifications
2. global importance of the lot to the supply chain or business
 - 2.1. priority, customer, development versus production
3. pacing lot movement
 - 3.1. fluctuation smoothing, flow balance cycle time allocation, delta schedule, critical ratio
4. local tool characteristics and performance
 - 4.1. batching and operational trains
 - 4.2. setup times
 - 4.3. parallelization opportunities
 - 4.4. differences in raw process time
 - 4.5. there may be ancillary equipment required at an operation in addition to the core toolset and labor

5. upstream and down stream requirements
 - 5.1. sending wafers to tools with limited work
 - 5.2. avoiding tools with large WIP piles
 - 5.3. balancing across rapidly repeating levels which use the same tool set

The core dispatch decision making activities can be divided into two primary components: guidance and judgment (figure 6.3).

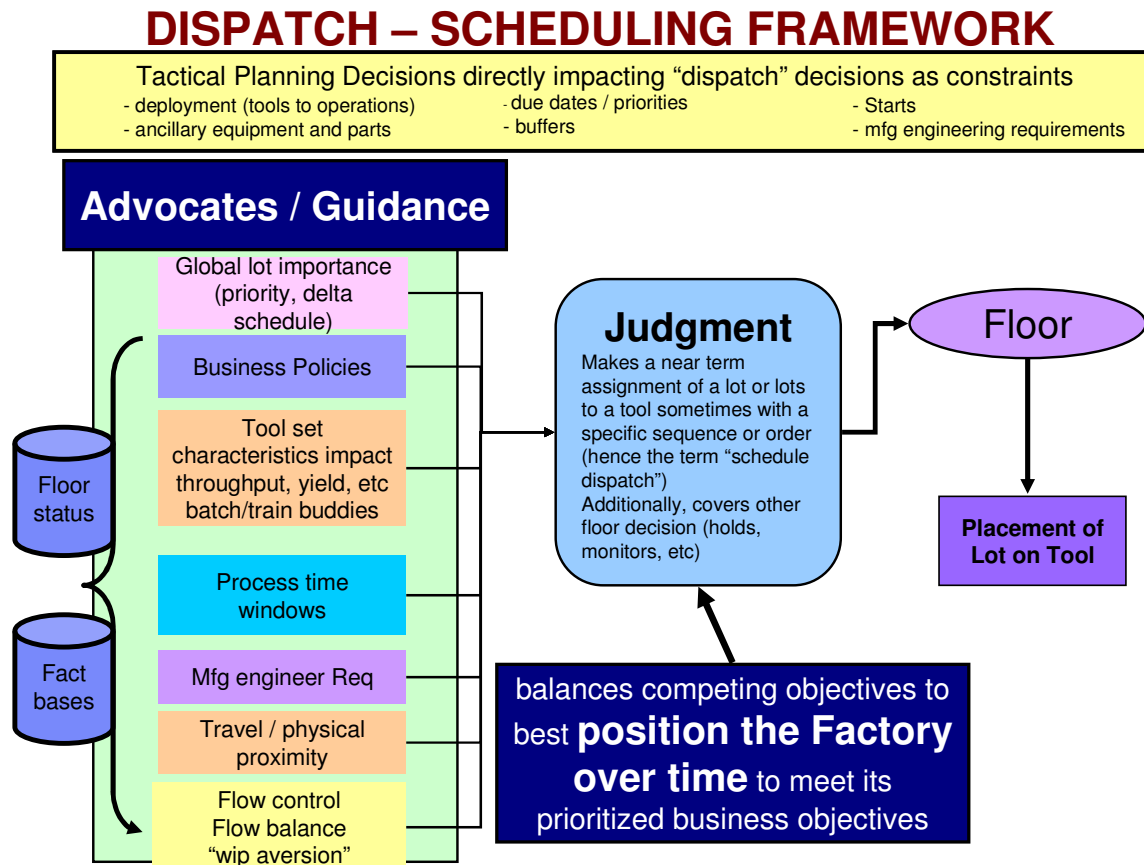


Figure 6.4: Dispatch Scheduling Framework

Guidance or advocate logic is the set of computational activities (which may be a program or manual) to create information posted to some location (often a table structure) that the assignment logic accesses or to trigger an assignment module to execute. The most common example is the calculation of some type of assessment whether a lot is ahead or behind of its pace. Another example is the updating of a fact base that may contain operation – tool preference based on static information (such as difference in raw process times between tools executing the same manufacturing action) or dynamic information (the amount of time will take to set the tool up to handle this manufacturing action). Other types of guidance include flow balance (avoid starving a tool set), manufacturing requirements (avoid running all lots of a certain type on a single tool, but distribute them across three tools), and process control time windows (lot must complete the next three steps within 5 hours or it will need to be scrapped due to contamination).

Judgment or assignment is the set of computational activities that when completed, result in a change of state or action on the manufacturing floor. The judgment logic must balance competing requirements such as meeting on time delivery, demand priorities, improving throughput with batches and trains, current WIP position, tool status, etc

The real goal of any judgment application is make a sequence of decisions over time that in aggregate improve the future position of the factory relative to its role in the total supply chain or demand supply network. The decisions are based on impact on the future state of the factory, not based on prior events. The sum total of the prior events has resulted in the current state of the factory. All other measures are attempts to create an interim goal that can be measured and decisions made against that is a reasonable approximation of the ultimate goal. Additionally, under some circumstances, these goals can be at odds with each other or the overall good of the demand supply network.

Simple Judgment typically

- does not consider assignment of lots to other tools in the tool group
- Does not consider the assignment of lots over time
- Does not consider upstream or downstream conditions (WIP level and Tool status and near term throughput rates)
- Applies simple rules of thumb for complex trade-off
- written with decision tree one pass logic
- generates a single decision, through a series of filters and if-then conditions
- Gives a reasonable (though myopic) decision
- relies on manual intervention for process time windows
- Typically have to be rewritten for different WIP levels (static adjustment)

Advanced Judgment typically

- looks across the tool set and upstream and downstream
- handles all process time windows
- establishes an anticipated sequence of assignments at all tools in a tool group over time for lots at the tool or which will arrive soon
- Measures the quality of the solution and anticipates impact on factory performance
- Uses an iterative search process in judgment logic
- dynamically adjusts for WIP levels and other business conditions
- easily handles lots that are intended to run at different speeds (typically given as a multiple of raw process time, referred to as CTM (cycle time multiplier) or XFactor.

6.3 Dispatch Challenges

The core challenge is to move from simple judgment to advanced judgment which requires transitioning the computational tools to make the judgment decision from “decision tree one pass logic”:

- join or link
 - sort and select
 - scoring
 - if then else
 - one pass logic no looping (except accumulation), algorithm
- to more advanced computational method that can handle the advanced data structures and solution search mechanisms needed to find smarter and more responsive solutions.

6.3.1 – Lots Traveling at Different Speeds

The increasing need for lots to travel at different speeds on the factory floor to meet different commitments has made simple dispatch rules such as FIFO (first in first out) and elapsed time impossible to use.

Why? Let's assume lot A03 needs travel at a speed of 3.8 (CTM) to meet its client commit date and lot A04 needs to travel at 3.3 to obtain the same result. A04 must travel faster than A03, which means it needs to absorb less wait time than A03. Inherent in the logic of "elapsed time" (lots that have been waiting the longest at a tool have the highest priority) is the wait time burden is the same for all lots, which is counter to the decision of the planners.

Assume for a moment A03 and A04 are both waiting to be processed at tool LION, A03 has been waiting at the tool for 185 hours and A04 has been waiting at the tool for 176 hours. Based on its longer elapsed time – A03 would be selected. However, if we look at the allocation of wait time burden, we see A04 at 176 is past its burden point of 175.7, while A03 at 180 is well below its burden point of 228.7.

Simple dispatch rules such as FIFO and elapsed time worked when factories made only a few products in large quantities with steady demand – a rare environment today. Therefore we observe factory responsiveness requires not just smarter planning, but the ability to execute which requires smarter dispatch.

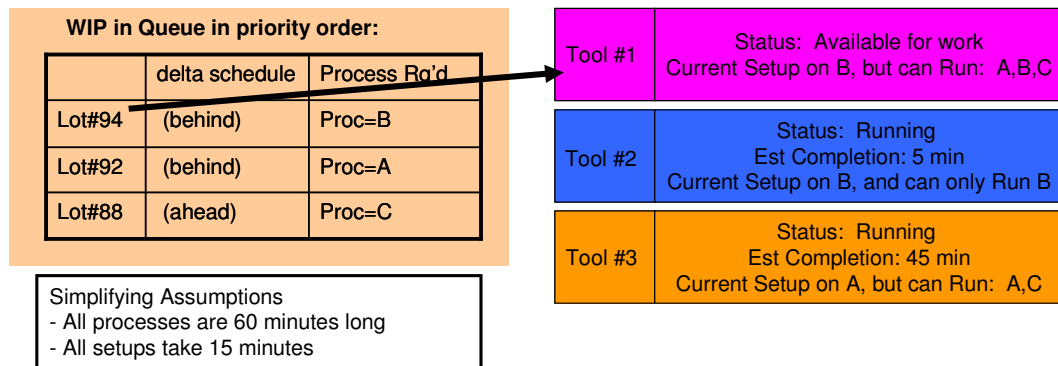
6.3.2 Looking Across a Tool Set and Building a Tentative Schedule

The following example clearly illustrates the limitations of rules for "dispatch heuristics" and the value in optimization methods to deploy dispatch scheduling. These limitations include: the inability to (a) look across time, (b) look across tools at a tool set, (c) create an anticipated sequence of events at a tool set over some time horizon, (d) establish a formal metric and (f) search alternatives.

In top half of [Figure 6.4](#) we have typical dispatch situation. There are three lots waiting to be processed. Two of the lots (94 and 92) are behind schedule. One of the lots is ahead of schedule (88). There are three tools in the tool group and their status is given in Figure 5. When tool#1 becomes available what lot is it assigned?

In bottom half of [Figure 6.4](#) we have typical "intelligent" assignment decision by a rules engine. The dispatch engine chooses lot #94 to tool #1 because the lot is behind schedule and the setup matches. This meets two key criteria: working on lots that behind and improving throughput by avoiding setups --- very sound thinking -- however?

As we play events forward in time, we quickly ([Figure 6.5](#)) find previously "undetected collateral damage" from our decision -- at 5 minutes: tool # 2 idle -> no wip available since tool #2 can only run process B and there are no lots in queue that require process

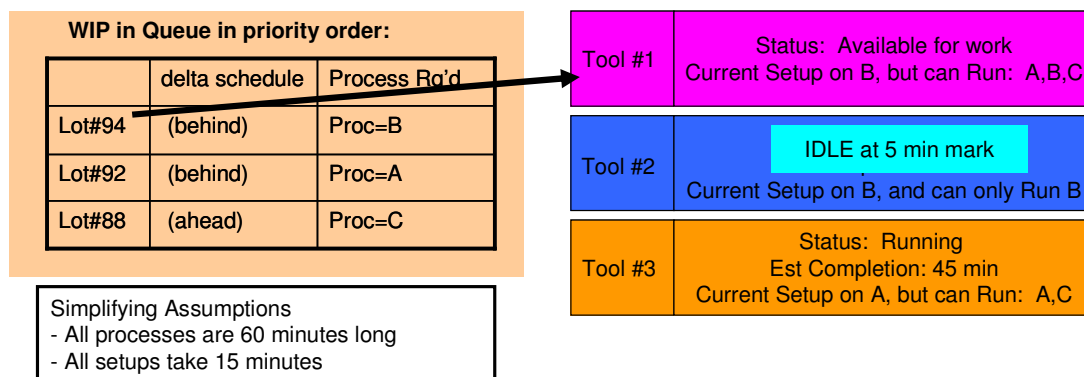


Tool #1 has just become available -> What should it run next?

Typically, heuristic dispatch will consider only the tool that is available. It will not look at the potential impact of actions in the future or across the tool set

At time 0: Tool#1 free, chooses lot #94 to tool #1 because the lot is behind schedule and the setup matches (sound thinking)

Figure 6.5: Example dispatch situation



Tool #1 has just become available -> What should it run next?

Typically, heuristic dispatch will consider only the tool that is available. It will not look at the potential impact of actions in the future or across the tool set

At time 0: Tool#1 free, chooses lot #94 to tool #1 because the lot is behind schedule and the setup matches

Undetected collateral impact of this decision

At 5 minutes: tool # 2 idle -> no wip available since tool #2 can only run process B and there are no lots in queue that require process B

At aggregate level shows idle with WIP (albeit wrong WIP)

Figure 6.6: Collateral damage and limits of dispatch

Let's return to the original decision and see how optimization can avoid the undetected collateral damage. In Figure 6.6 we see three possible solutions that are projected forward through the complete of all lots currently waiting to be serviced by this tool group. Observe for each potential solution a tentative detailed schedule (in our example, we round to 10 minute intervals simply to avoid overburdening the reader with detail, but FPO keeps details to the minute or less if needed). Since we are projected forward in time across lots and across tools we can directly measure the impact of the decision in terms of such items as completion time for lots (Figure 6.6) and tool utilization or idle time (Figure 6.7). For both measurements, options 2 and 3 substantially

Three Options

scaled to 10
min intervals,
start/stop are
approximate

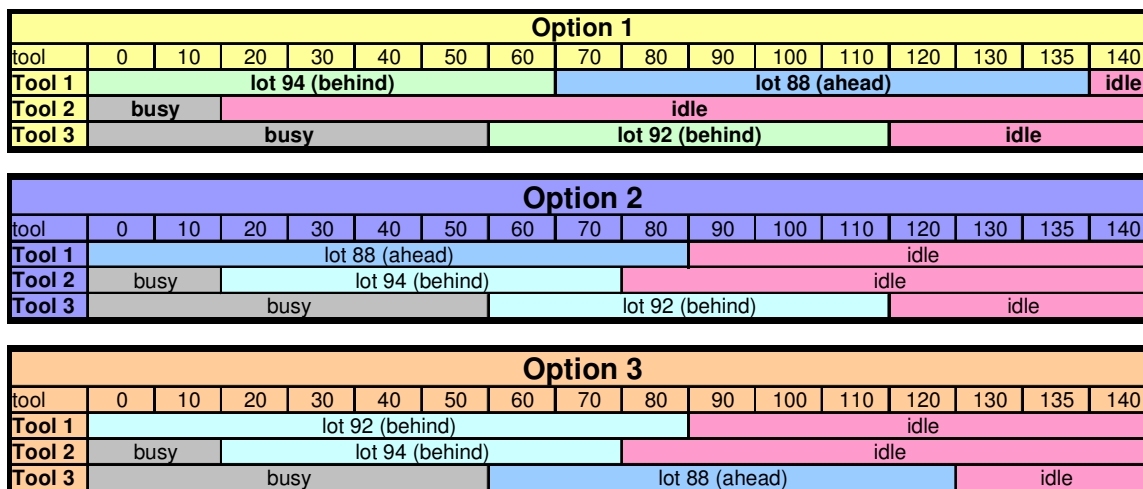


Figure 6.7: Three sequence schedule options

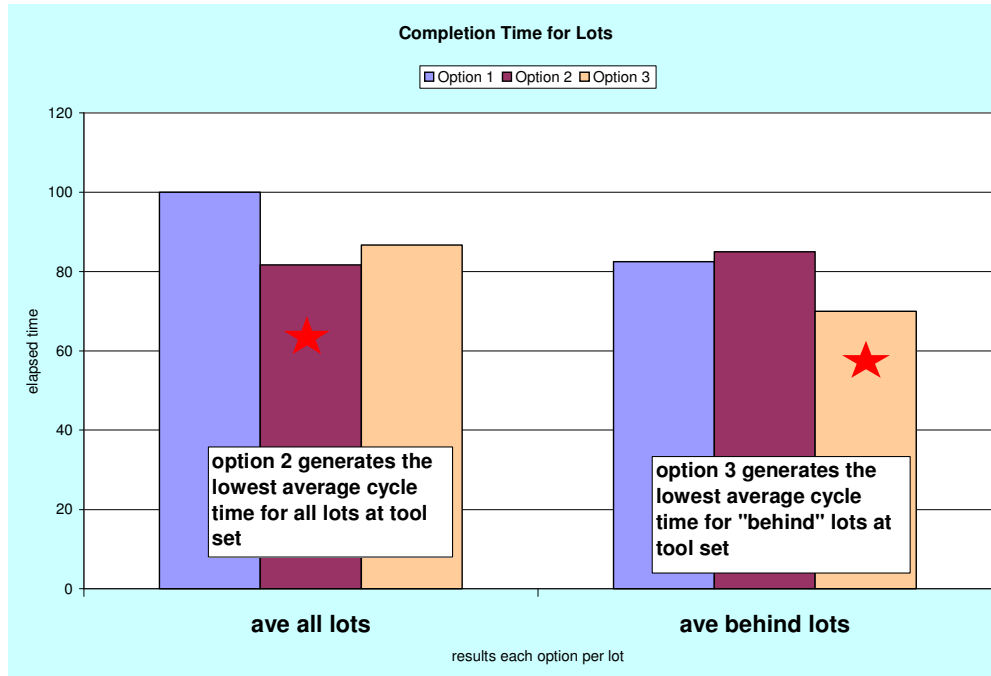


Figure 6/8: Comparison of three options for lot complete time

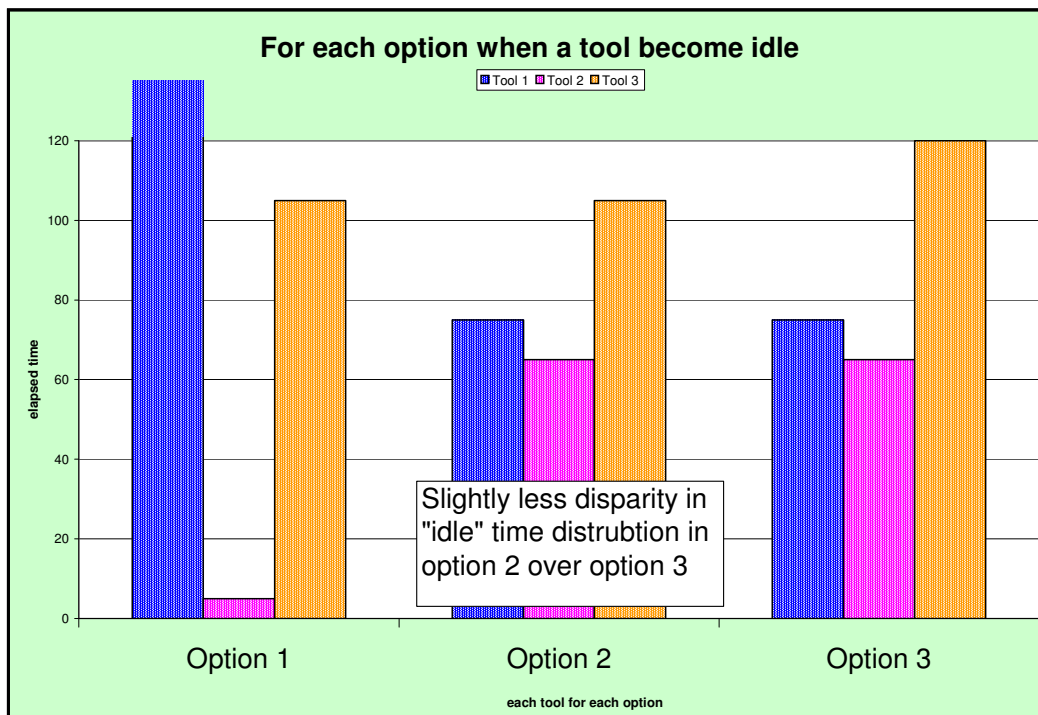


Figure 6.9: Comparison of three options for tool utilization

The application of advanced computational methods improved the assignment decision by generating optional decisions across time and tools, evaluating these decisions with a firm metric, searching for other options, and having a selection or stopping criteria. Observe: Option 2: lower average cycle time for lots and less variation in exit time for lots and tool use. Option 3: lot#92 (which is behind) completes sooner good utilization of tools. Both options are superior to option 1. The best option depends on (a)details (how far behind and ahead, lot priority, demand class, (b)near term WIP conditions, and (c)business priorities. However the value of optimization goes well beyond a better assignment decision, but provides a visibility impossible to obtain with simple point of sale rules. It enables everyone to clearly see an anticipated projected schedule for lots and tools and anticipated tool utilization and WIP levels. This is critical, since the assignment decision is strongly influenced by many other decisions – two key ones are deployment decisions (the processes a tool is qualified and/or allowed to handle) and the movement, location, and cleaning of masks or reticles.

6.3.3 Examples of Limits to Rules to Specify Desired Dispatch Decision

A typical goal for a fab is to eliminate unnecessary transport of lots. The goal is “minimize” lot movement and this part of a goal of minimizing cycle time. The optimization engine with create solutions with this goal in mind.

A rule engine requires the rule writer to specify “how to limit” travel. A typical rule would be: <If lot is of type A, B, or C do not let it travel to another sector to be processed by a tool until it has waited at least 30 minutes in its current sector>. What the rule really wants is to keep a lot from traveling to another sector from its current sector if there is a tool in this sector that can handle the lot reasonably soon. There is no way to specify this general policy. So the rule specifies the “specifics” of how and hopes many times a tool will become free in the lot’s current sector within 30 minutes that is a good match for this lot. Obviously, if a fab is heavily loaded this is a higher probability event. A fab that is only moderately loaded could have a problem and the rule would need to be manually changed.

A second common limitation involves the use of rankings. Assume lots A, B, and C are behind schedule by 1.1, 1.2, and 4.8 days respectively. The lot C has the rank of 1 (most behind) since it is 3.6 days worse off the lot B. Alternatively if lot C was behind schedule by 1.3 days it would still have the rank of 1, but it just barely worse off (0.1 days) then lot B. Typically, rule based dispatch will work with the ranking and not be able make use of the richer information about the magnitude of the difference.

7. Interaction between Central Planning, Factory Planning, and Dispatch

7.1 Factory Limits Responsiveness - Opportunities Abound

The primary steps in the central planning process are

1. Create a demand statement
2. Capture the flow of materials in the demand supply network
3. Gather and collect key information from the factory
 - 3.1. Project the completion of WIP to a decision point (often completion of the part).
 - 3.2. a statement of capacity required and available
 - 3.3. a statement of lead time or cycle time to complete a new start
4. Create a model that captures key enterprise relationships (Central Planning Engine – CPE)
5. Create an enterprise wide central plan by matching current and future assets with current and future demand using the CPE to create a future projected state of the enterprise and the ability to soft peg the current position of the enterprise to the projected future position. Information from the model includes
 - 5.1. a projected supply linked with exit demand
 - 5.2. identification of at risk orders either to a commit date or request date

5.3. Synchronization signals across the enterprise

5.4. Capacity utilization levels

Steps 4 and 5 in the central engine planning process (CPE) are often viewed as the planning “hub” and the focus of making a firm more responsive through “smarter” engines and better (quality and timeliness) data. However, in Step 3 the factory sets the boundaries of “responsiveness”. The CPE relies on the factory to provide

1. an estimated completion date for each lot in the line (either to completion or staging point)
2. a statement of capacity available and required for each manufacturing start (typically at an aggregate level)
3. an estimated lead or cycle time to complete a start fixed for some time interval

Additionally,

4. The central planning process can not change the due date on the lot or the lot’s priority without extensive manual negotiations with manufacturing
5. Central planning has no control (and typically no knowledge) of the lot importance metric used by the factory or how it balances utilization and delivery.
6. Each piece of information supplied by the factory to the central planning process is “fixed” – stripped of all of the information that enables trade-offs to be made. For example
 - 6.1. slowing one lot down to enable another lot to go faster
 - 6.2. trade-offs between cycle time and capacity available based on the operating curve
 - 6.3. redeploying tools to handle a different mix of manufacturing processes or products

Culturally, factories “dislike” change. Factories “like” steady rates of production referred to as smooth flow. This has been reinforced with various lean initiatives. Despite the substantial forces to limit change, constant pressure from emerging market opportunities to manufacturing excursions to inaccuracies in planning (deviations between the plan and the actual) drive an ongoing sequence “off line one of a kind” negotiations between the central planner and the factory planner to make adjustments that rely on quasi-manual decision support tools with limited function. For example

1. A client may need three lots four days earlier than committed and this can be accommodated simply by placing these lots on expedite.
2. The demand for product A requires 30 units of capacity of Tool Set A1 on average each day. Tool Set A1 only has 25 units of capacity available. Tool Set A2, which is not listed as a capacity option for product A, can service product A, but it runs much slower. A review of capacity utilization for Tool Set A2 indicates it will be underutilized. A decision is made to qualify Tool Set A2 to handle product A.
3. A client has had a steady order for 10 units daily of product W with a cycle time of 15 days where the key tool set is Tool Set W3. The business has been able to achieve an on time delivery rate of 97%. The client would like to increase its standing order from 10 units to 12 units. Central planning initially rejects this opportunity since the stated maximum capacity in their model for Tool Set W3 is 10. However, when the two planners look at the details of the tool set and its operating curve, the business decides it can commit to 12 per day if the cycle time is increased to 16 days (or if the OTD commit percentage is lowered).
4. Assume a client has placed an order for 5 lots of “part A” per day with a cycle time of 10 days. On average there are 50 lots of “part A” in WIP and the factory completes 5 lots per day. The due date posted on each lot is the start date plus 10 days. For example, lots started on day 6 are due on day 16. Due dates on the lots can only be changed manually by a factory planner. The factory has an abnormal set of tool outages and goes 3 days without delivering any lots – it is past due 15 lots (3x5). It has continued to start 5 lots per day. At the start of the fourth day the number of lots in the line is 65 (50 + 15 past due). On the evening of the third day the client and central planning huddle about a recovery strategy. The client determines demand has been soft for this part and agrees to “forgive” 5 lots and have the remaining 10 lots “caught” up at a pace of 1 per day (in addition to the regular 5). Therefore the new order book for this client is 6 lots per day for the next 10

days and then returns to 5 per day. Without changes to the due date on the lots in WIP, the factory continues to see 15 lots past due and will drive to “catch up” as quickly as possible. The factory may decide to delay lots for a second client to catch up all 15 past due lots in 5 days. Therefore the factory planner has to manually change the due dates on the lots to insure the factory floor has the correct guidance.

These examples make clear when central planning can make effective use of the flexibility within the factory that is hidden from its traditional view – good things can happen. The opportunity for improved responsiveness simply needs to “widen and straighten” this trail with appropriate planning and dispatch tools, processes, and protocols. Each one would be considered muda (wasteful) by lean and the correct response is to eliminate them, not to build tools to make doing this more intelligent and efficient. We contend the goal of any firm is to eliminate unnecessary complexity, but ignoring the complexity that remains is like tackling snow storms in the northeast with bald tires.

Additionally, such tools and processes can keep bad things from happening. For example, if many lots are being “expedited” already there is no room for one more lot. Tool W3 may be needed for engineering lots not in the central planning data or it may have a history of time consuming qualifications making it too large a risk. Just as the factory prefers “steady” and conservative, central planning often fall preys to an overly optimistic mindset that is fine with constant churn. Tools for improving factory/enterprise coordination fall into three groups

1. direct interaction with central planning tools (for example WIP projection, expedite decisions, demand pegging, specialized capacity planning models for flexibility in manufacturing)
2. tactical decision models (tool deployment, allocation of cycle time or moves)
3. dispatch scheduling (assigning lots to a tool)

The following sections provide some examples.

7.2. Committing Some Lots to Run a little Faster – Collateral Impact

A common, yet manual, practice is for central planning to negotiate with factory planning to “speed” up certain lots to meet a customer request, overcome a manufacturing delay, or compensate for a planning failure. Typically, the analysis is limited and ad hoc with no comprehensive process as seen in central planning or tool planning. There are rules of thumb such as:

- number of “expedite” lots can not exceed some fixed number N or percentage of total lots
- The fastest an expedite lot can run is some CTM less than the normal lots. If normal lots have a CTM of 5, expedite lots might have a CTM of 3.

A closer look makes it clear the core of this decision is a reallocation of either wait time or factory moves that enables some lots to run faster by having others run slower over some subset of the manufacturing line over some time duration. For example, assume the factory has five lots (LOT01 ... LOT05) in the last stage of production; each lot requires 4 moves (manufacturing actions) to complete; the maximum number of total moves (capacity) per day is 5; and the most moves a lot can have in one day is 2. If capacity is allocated “fairly,” then each lot gets one move per day and each lot finishes in 4 days. Now assume the business decides LOT01 and LOT02 must finish in two days, then each needs 2 moves per days for two days, and therefore on each of these days two lots sit “idle” during these two days to enable this expedite.

Fordyce and Milne [18] develop this allocation concept in more detail focusing on wait time allocation instead of moves. In each case lots that look essentially the same are required to run at different speeds on the factory floor. The planned speedup is worthless without successful factory execution. This places a substantial burden on dispatch and precludes the use of simple methods (and lean favorites) such as FIFO (first in first out) and elapsed time. Again we see that lean and responsiveness are not in sync.

This topic is part of an area called *General Plan Repair Process*. In this process central and factory planners look to identify actions to take that will enable orders that are currently flagged as “late” to be met on time.

7.3 smarter central planning through better modeling of factory capacity

As we outlined before, the central planning process requires as critical inputs from the factory: capacity (required and available) and cycle times. Since the 1980s manufacturing resource planning (MRP) and material balance equations (MBE) in optimization formulations have been the two dominant methods used in central planning [74]. In these methods the factory representation is “static” and linear. The cycle times and capacity information are fixed across some time period and handled with linear relationships.

Historically intricacies of factory tool planning (availability, deployment decisions, cascading, setup times, batching, et al) and the dynamic interaction between equipment utilization (effective capacity) and cycle time through the operating curve have for the most part been ignored. This will not be sustainable in the future as the burden on responsiveness resulting in under utilization or delivering products late is increasingly unacceptable.

In the previous section we described a situation where the client needed 30 units per day and the initial central planning analysis determined the maximum the factory could make was 25. The following section elaborates on the method to improve responsiveness by capturing alternative deployments of tools to manufacturing operations.

7.3.1 Revisiting Capacity Allocation – a Rabbit Out of the Hat

The core of resource allocation in central planning engines (CPE) is linking a manufacturing activity to one more resources; establishing a consumption rate for each unit of production by that manufacturing activity for the selected resource; and providing the total available capacity for the resource. In **Table 7.1**, we see operations 101 and 151 can be handled by Tool A or B. Operations 201, 202, 301, and 302 can be handled only by Tool A. **Table 7.2** tells us the available capacity for Tool A and Tool B is 1152 working minutes per time unit (for example per day).

Table 7.1: Operation Resource Linkage		
Manufacturing Activity	Resource	Consumption Rate
Operation 101	Tool A	10
Operation 101	Tool B	10
Operation 151	Tool A	10
Operation 151	Tool B	10
Operation 201	Tool A	15
Operation 202	Tool A	15
Operation 301	Tool A	15
Operation 302	Tool A	15

Table 7.2: Available Capacity	
Resource	Consumption Rate
Tool A	1152
Tool B	1152

Assume, we have a uniform start rate of one lot per day and each lot goes through each operation (101, 151, 201, 202, 301, and 302) once. In steady state each operation would need to process one lot per day. The optimal way to allocate operations to tools is assign operations 101 and 151 to Tool B and the remaining four operations (201, 202, 301, and 302) assigned to Tool A. This creates a load on Tool B of 20 ($=10 + 10$) minutes of processing and 60 ($=15+15+15+15$) minutes on Tool A. Since Tool A has 1152 units available, the maximum number of lots per day is 19.2 ($=1152/60$)

If the demand rises to 30 lots, the CPE would indicate this was not feasible and mostly likely push some production out in time showing the pieces being delivered late.

However, the CPE lacks access to the tactical deployment detailed information and therefore has no method to identify better solutions. If the customer is important and his

demand projected to be delivered late, the central planner will contact the factory planner and decisions and review of the detailed deployment decision will occur to “mine for capacity” by looking for opportunities to reallocate capacity to satisfy the demand on time. In many industries a tool can potentially handle many different operations, but at a given point in time is only actively deployed (linked) to a small subset of these operations.

This “reduced” deployment occurs for a number of reasons including:

1. It is physically impossible for the tool to be “actively” ready for more than small number of operations. If we want the tool to handle an operation different then those currently selected, the tool has to be brought down for a while, “reconfigured,” and brought back up.
2. There are manufacturing performance advantages to limiting the number of operations a tool is currently deployed to handle.
3. The manufacturing team often uses deployment decision to attempt to “balance” tool load by estimating future workload
4. The manufacturing team prefers to deploy its fastest tools to certain operations to keep total cycle time low.
5. Habit or prior practice

The reality is the tactical deployment decision made by manufacturing when reflected in the capacity information sent to central planning understates the flexibility of manufacturing to produce parts to meet an increase in demand. This flexibility can only be uncovered through manual intervention when central planning presses factory planning.

In our example, it might well be Tool C can, after being retooled, be switched from working on “gadgets” to “widgets” --- specifically it could be re-configured to handle operations 201 and 301. This might be reflected in the following updated **tables (7.3 and 7.4)**. To add to the complexity, Tool C still needs to be used for some of the gadget operations.

Table 7.3: Operation Resource Linkage		
Manufacturing Activity	Resource	Consumption Rate
Operation 101	Tool A	10
Operation 101	Tool B	10
Operation 151	Tool A	10
Operation 151	Tool B	10
Operation 201	Tool A	15
Operation 201	Tool C	30
Operation 202	Tool A	15
Operation 301	Tool A	15
Operation 301	Tool C	30
Operation 302	Tool A	15

Table 7.4: Available Capacity	
Resource	Consumption Rate
Tool A	1152
Tool B	1152
Tool C	2000

Additionally, the model in the central planning engine can be enhanced to accommodate:

1. Tool – operation preferences and costs (only assign operation 301 to Tool C as a last resort)
2. Minimum tool usage (if operation 301 is assign to Tool C then it needs least 2 units of work per day to stay qualified)
3. Limitations of tool-operation pairings (Tool C can do work from Operation 301 or 201, but not both)

We might be tempted to state the old adage – garbage in garbage out – in fact that would be wrong and fail to understand the environment that generated the “limited” but accurate capacity information. In fact, the capacity information provided by manufacturing to the central supply chain model as reflected in Tables 1 and 2 was accurate, but limited. It was limited to the current near term production requirements and the near term ability to use the tools. Tool C can not be used for operations 201 and 301 until it is “retooled” and qualified. This may take a day or may take a week – but certainly manufacturing can not afford to “retool” on a daily basis.

8. Conclusion: New Science Emerges and Extends the Borders of Bounded Rationality

When one of the authors, Ken Fordyce, joined IBM in 1977 as a junior programmer with an undergraduate degree in mathematics, a few courses in operations research and statistics, one of the first people he met was Herschel Smith. Herschel was in his 60s and Ken was 23. Herschel had built a small linear programming model to optimize taxes for IBM World Trade. His efforts were recognized and appreciated by IBM, but he was part of an Information Systems organization. At that time, modeling did not have enough traction to be its “own person,” and decision modeling applied to business problems was at most 25 years old.

Today some firms use combination of linear programming, queuing networks, and other advanced computational methods to do central planning, factory planning, and dispatch. These firms cannot think of “life without these decision models.” We believe we are witnessing a growing awareness of the importance of decision models and competing on analytics [9]. What computational decision scientists offer is the potential to “Extend the Borders of Bounded Rationality.”

Herbert Simon (Nobel Prize Winner in Economics) observed, “as humans, we have ‘bounded rationality’ and break complex systems into small manageable pieces.” The challenge for organizations is to integrate information and decision technology to push boundaries out and improve performance. Nick Donofrio (IBM Senior Vice President) observed, “Access to computational capability will enable us to model things that would never have believed before.” The challenge reaches beyond coding algorithms, linking to data, and turning it on. Each decision-science team must execute its role as “intelligent evolutionists” to ensure the organization adopts complex decision technology in a sustained incremental fashion.

This will enable organizations from semiconductor firms to hospitals to be more responsive.

“The ability to simultaneously respond to customers’ needs and emerging business opportunities in an intelligent orderly manner is a survival requirement for today’s market place. Our customers continue to tell us that the quality of our responsiveness is as important as the quality of our products ... The work done by the innovators on our PROFIT team, as well as their colleagues, has enabled us to meet the customer responsiveness challenge. And I’m convinced that the monetary savings associated with the PROFIT work [\$80 million in 1999 alone] – though significant in their own right – understate the work’s true value to IBM.” (Nick Donofrio, IBM Senior Vice President [60])

That said, there is still a long way to go and in many respects we only have the illusion of control (**Kempf [51]**). People are comfortable with their guesses and decision scientists often fail to deliver real value. Politely, life is much cleaner if your modeling work remains an academic exercise. The biggest obstacle is may not be technology, but social order.

“There is no more delicate matter to take in hand, nor more dangerous to conduct, nor more doubtful of success, than to step up as a leader in the introduction of changes. For he who innovates will have for his enemies all those who are well off under the existing order of things, and only lukewarm support in those who might be better off under the new.” (Niccolo Machiavelli)

Bibliography

1. Bermon, S. and Hood, S. (1999), “Capacity Optimization Planning System (CAPS),” *Interfaces*, Vol. 29, No. 5, pp. 31 – 50.
2. Bitran, G. and Tirupati, D. (1989), “Tradeoff Curves, Targeting and Balancing in Manufacturing Networks,” *Operations Research*, Vol. 37, No. 4, pp. 547- .555
3. Bixby, R., R. Burda, and D. Miller. 2006. Short-interval detailed production scheduling in 300mm semiconductor manufacturing using mixed integer and constraint programming” in Semiconductor fabtech – 32nd edition, <WWW.FABTECH.ORG,> 34-40

4. Chen, H., Harrison, M., Mandelbaum, A., Ackere, A., and Wein, L. (1988), "Empirical Evaluation of a Queuing Network Model for Semiconductor Wafer Fabrication," *Operations Research*, Vol. 36, No. 2, pp. 202-215.
5. Connors, D., Feigin, G., and Yao, D., 1996, "A queuing network model for semiconductor manufacturing", *IEEE Transactions on Semiconductor Manufacturing*, 9 (3), 412-427
6. Consilium, 1988. Rule Based Dispatch User's Guide, Mountain View, CA.
7. Dabbas, R. and J. Fowler 2003. A New Scheduling Approach Using Combined Dispatching Criteria in Wafer Fabs, *IEEE Transactions on Semiconductor Manufacturing*, 16:3 501-510
8. Dangat, G.S., Milne, R.J., and Orzell, R.A. (2000), "Method to Provide Common Support for Multiple Types of Solvers for Matching Assets with Demand in Microelectronics Manufacturing," U.S. Patent 6,041,267 (3/21/2000).
9. Davenport, T. (2006), "Competing on Analytics," Harvard Business Review, January 2006, pp. 1 – 9.
10. Dennis, P. (2007), *Lean Production Simplified*, Productivity Press, NY
11. Denton, B., Hedge, S., and Orzell, R.A. (2004), "Method of Calculating Low Level Codes for Considering Capacities," U.S. Patent 6,584,370 (5/18/2004).
12. Denton, B., Forrest, J., and Milne, R.J. (2005), "A Method for Considering Hierarchical Preemptive Demand Priorities in a Supply Chain Optimization Model," U.S. Patent Application: 2005-0171828, also, IBM docket: BUR9-2003-0198US1.
13. Denton, B. and Milne, R.J. (2006a), "Method for Optimizing Material Substitutions within a Supply Chain," U.S. Patent 6,983,190 (1/3/2006)
14. Denton, B., Forrest, J., and Milne, R.J. (2006), "Methods for Solving a Mixed Integer Program for Semiconductor Supply Chain Optimization at IBM," *Interfaces*, Vol. 36, No. 5, September – October 2006, pp. 386 – 399
15. Fargher, H. and R. Smith, 1994. Planning in a Flexible Semiconductor Manufacturing Environment, Chapter 19 in *Intelligent Scheduling*, edited by Mark Fox and Monte Zweben, Morgan Kaufman Publishers
16. Fordyce, K., Bixby, R. and Burda, R. (2008), "Technology That Upsets The Social Order – A Paradigm Shift In Assigning Lots To Tools In A Wafer Fabricator – The Transition From Rules To Optimization," *Proceedings of the 2008 Winter Simulation Conference*
17. Fordyce, K., Wang, C-T, Chang, C., Degbotse, A., Denton, B., Lyon, P., Milne, R.J., Orzell, R., Rice, R., and Waite, J. (2011a), "The Ongoing Challenge—Creating an Enterprise-wide Detailed Supply Chain Plan for Semiconductor and Package Operations," chapter 14 in *Planning Production and Inventories in the Extended Enterprise: A State of the Art Handbook, Volume 2*, edited by Kempf, Keskinocak, and Uzsoy
18. Fordyce, K., Milne, R.J., et al (2011b), "Improving Factory Responsiveness—an Optimization Model for Wait Time Allocation," working paper, fordyce@us.ibm.com, jmilne@clarkson.edu
19. Fordyce, K., Milne, R.J., et al (2011c), "Basics of the Operating Curve – Classical Planning Meets Its Uncertainty Principle," working paper, fordyce@us.ibm.com, jmilne@clarkson.edu
20. Fordyce, K. Milne, R.J., et al (2011d), "Dynamically Linking Cycle Time Decision with Capacity Required Inputs in Central Planning Models," working paper, fordyce@us.ibm.com, jmilne@clarkson.edu
21. Fordyce, K. Milne, R.J., et al (2011e), "Using Optimization Methods to Help with Deployment Decisions in Factory Planning," working paper, fordyce@us.ibm.com, jmilne@clarkson.edu
22. Fowler, J. W., Brown, S., Gold, H., and Schoemig, A., 1997, "Measurable improvements in cycle-time-constrained capacity", *Proceedings of the 6th IEEE/UCS/SEMI International Symposium on Semiconductor Manufacturing (ISSM)*, San Francisco, CA
24. Fowler, J. W. and Rose, O., 2004, "Grand challenges in modeling and simulation of complex manufacturing systems," *Simulation*, vol. 80, no. 9, pp. 469–476, 2004
25. Fox, B. and Kempf, K (1985), Complexity, Uncertainty, and Opportunistic Scheduling,

- Proceedings of the IEEE Second Conference on Artificial Intelligence Applications: The Engineering Knowledge Based Systems, Miami, Florida, 487-492.
26. Fox, M. 1987. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*, Morgan Kaufman Publishers, Los Altos, Ca.
 27. Galbraith, J. (1973), *Designing Complex Organizations*, Addison-Wesley.
 28. Glassey, C. et al, 1996. Control Rules for Production Control Semiconductor Fabs, *IEEE Transactions on Semiconductor Manufacturing*, 9:4 536-549.
 29. Glassey, C. and W. Weng 1991. Dynamic Batching Heuristic for Simultaneous Processing, *IEEE Transactions on Semiconductor Manufacturing*, 4:2 77-82.
 30. Goldman, S. (2004), "Science in the Twentieth Century," Great Courses on CD by the Teaching Company, Chantilly, VA.
 31. Glover, F., Jones, G., Karney, D., Klingman, D., and Mote, J. (1979), "An Integrated Production, Distribution, and Inventory Planning System," *Interfaces*, Vol. 9, No. 5, pp. 21 – 35.
 32. Govind, N. et al 2007, "Integrated Targeting, Near Real-Time Scheduling, and Dispatching with Automated Execution in Semiconductor Manufacturing", *2007 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 87-92.
 33. Graves, S., H. Meal, D. Stefek, and A. Zeghmi, 1983. .Scheduling of Re-entrant Flow Shops, *Journal of Operations Management*, 3: 197-203
 34. Graves, R.J., Konopka, J.M., and Milne, R.J. (1995), "Literature Review of Material Flow Control Mechanisms," *Production Planning and Control*, Vol. 6, No. 5, 395 – 403.
 35. Gurnani, H., R. Anupindi, and R. Akella 1992. Control of Batch Processing Systems in Semiconductor Wafer Fabrication Facilities, *IEEE Transactions on Semiconductor Manufacturing*, 5:4. 319-327
 36. Hackman, S.T. and Leachman, R.C. (1989), "A General Framework for Modeling Production," *Management Science*, Vol. 35, No. 4, pp. 478 – 495.
 37. Hegde, S.R., Milne, R.J., Orzell, R.A., Pati, M.C., and Patil, S.P. (2004), "Decomposition System and Method for Solving a Large-Scale Semiconductor Production Planning Problem," United States Patent No. 6,701,201 B2.
 38. Hopp, W. and Spearman, M. (2008), *Factory Physics, 3rd Edition*, McGraw-Hill, NY
 39. Horn, G. and Podgorski, W. (1998), "A focus on cycle time-vs.-tool utilization "paradox" with material," *Advanced Semiconductor Manufacturing Conference and Workshop Proceedings*, pp. 405 – 412
 40. IBM white paper G299-0906-00 (2006), "Collaboration with IBM E&TS Helps ADI Stay ahead of Customer Demand," IBM Corporation, 1133 Westchester Avenue, White Plains, NY 10604, U.S.A.
 41. Ignizio, J. 2009, *Optimizing Factory Performance*, McGraw-Hill, NY
 42. Johri, P. 1989, Dispatching in an Integrated Circuit Wafer Fabrication Line, *Proceedings, Winter Simulation Conference*, 918-921.
 43. Johri, P. 1993, "Practical issues in scheduling and dispatch", *Journal of Manufacturing*
 44. *Systems*, 12 (6), 474-485
 45. Karmarkara, U., 1993 "Manufacturing lead times, order release and capacity loading", chapter 6 in *Handbooks in Operations Research and Management Science Vol. 4 – Logistics of Production and Inventory*, edited Henderson and Nelson, Pages 287-329
 46. Kempf, K. 1989, Manufacturing Scheduling: Intelligently Combining Existing Methods, in Working Notes of AAAI AI in Manufacturing Symposium, M. Fox editor, AAAI, 445 Burgess Drive Menlo Park, CA 94025-3496.
 47. Kempf, K. 1989., Manufacturing Planning and Scheduling: Where We Are and Where We Need to Be, *Proceedings of the Fifth IEEE Conference on AI Applications*, IEEE Computer Society Press, Los Alamitos, CA. 13-19.
 48. Kempf, K, Pape,D., Smith, S and Fox, B (1991). Issues in the Design of AI Based Schedulers, *AI Magazine*, 11:5, 37-45.

49. Kempf, K (1994), "Intelligent Scheduling Semiconductor Wafer Fabrication," Chapter 18 in *Intelligent Scheduling*, edited by Mark Fox and Monte Zweben, Morgan Kaufman Publishers, 473-516.
50. Kempf, K. (2004), "Control-Oriented Approaches to Supply Chain Management in Semiconductor Manufacturing," *Proceedings of the 2004 American Control Conference*, Boston, MA, pp. 4563 – 4576.
51. Kempf, K. 2007, "Complexity and the Enterprise – The Illusion of Control", Intel Corp,
52. Leachman, R., Benson, R., Liu, C., and Raar, D. (1996), "IMPreSS: An Automated Production Planning and Delivery-Quotation System at Harris Corporation - Semiconductor Sector," *Interfaces*, Vol. 26, No. 1, pp. 6 – 37.
53. Lee C., R. Uzsoy, L. Martin-Vega and P. Leonard, 1991. Production Scheduling Algorithms for a Semiconductor Test Facility, *IEEE Transactions on Semiconductor Manufacturing*, 4: 271-280.
54. Lee C., R. Uzsoy, and L. Martin-Vega 1992, Efficient Algorithms for Scheduling Semiconductor Burn-In Operations, *Operations Research*, 40:4, 764-775.
55. Lin, G.1992. "An Opportunistic Price-Based Multiple Resource and Part Scheduling," Ph.D. dissertation, School of Industrial Engineering, Purdue University,
56. Lin, G., Ettl, M., Buckley, S., Yao, D., Naccarato, B., Allan, R., Kim, K., Koenig, L. (2000), "Extended Enterprise Supply Chain Management at IBM Personal Systems Group and Other Divisions," *Interfaces*, 30(1), pp. 7 – 25.
57. Little, J. (1970), "Models and Managers: The Concept of a Decision Calculus," *Management Science*, Vol. 16, No. 8, pp. B-466 – B-485.
58. Little, J. (1992), "Tautologies, Models and Theories: Can We Find "Laws" of Manufacturing?," *IIE Transactions*, Vol. 24, No. 3, pp. 7 – 13.
59. Liu, J., Yang, F., Wan, H., and Fowler, J. (2010), "Capacity Planning Through Queuing Analysis and Simulation-based Statistical Methods: a Case Study for Semiconductor Wafer FABs," web.ics.purdue.edu/~hwan/docs/IJPR
60. Lyon, P., Milne, R.J., Orzell, R., Rice, R. (2001), "Matching Assets with Demand in Supply-Chain Management at IBM Microelectronics," *Interfaces*, 31:1, pp. 108 – 124.
61. Orzell, R., Patil, S., and Wang, C. (2004) "Method for Identifying Product Assets in a Supply Chain Used to Satisfy Multiple Customer Demands," U.S. Patent 20050177465A1 (10/17/2004).
62. Martin-Vega. L. et al, 1989, "Apply Just in Time at a Wafer Fab case study", *IEEE Transactions on Semiconductor*, Vol. 2, No. 1,16-22.
63. Martin, D. (1999), "Capacity and cycle time-throughput understanding system (CACSUS)," *Advanced Semiconductor Manufacturing Conference and Workshop Proceedings*, pp. 127 - 131
64. Morrison, J. and Martin, D. (2006), "Cycle Time Approximations for the G/G/m Queue Subject to Server Failures and Cycle Time Offsets with Applications," *ASMC 2006 Proceedings*, pp. 322
65. Orlicky, J. (1975), *Material Requirements Planning: The New Way of Life in Production and Inventory Management*, McGraw-Hill, New York.
66. Pogge, R. (2009), "Real World Relativity", www.astronomy.ohio-state.edu/~pogge/Ast162/Unit5/gps.html
67. Robinson, J. K., Fowler, J. W. and Bard, J. F., 1995, "The use of upstream and downstream information in scheduling semiconductor batch operations", *International Journal of Production Research*, 33 (7), 1849-1870
68. Savell, D., R. Perez, and S. Koh, 1989. Scheduling semiconductor wafer production: an expert system implementation", *IEEE Expert* 4:3 9-15
69. Shirodkar, S. and Kempf, K (2006), "Supply Chain Collaboration Through Shared Capacity Models", *Interfaces*, Vol. 36, No. 5, pp. 420-432
70. Shobrys, D. (2003), "History of APS," Supply Chain Consultants (www.supplychain.com), 460 Fairmont Drive, Wilmington, DE 19808, U.S.A.

73. Simon, H.A. (1957), *Administrative Behavior*, Second Edition, The Free Press, New York.
74. Singh, H. (2009a), *Supply Chain Planning in the Process Industry*, Supply Chain Consultants, Wilmington, De.
75. Singh, H. (2009b), *Practical Guide for Improving Sales and Operations Planning*, Supply Chain Consultants, Wilmington, De.
76. Singh, H. (2010), "Effectively Balancing Demand and Supply Forecasting in a Rebounding Economy," Supply Chain Consultants, Wilmington, DE.
77. Sullivan, G. (1990), "IBM Burlington's Logistics Management System (LMS)," *Interfaces*, Vol. 20, No. 1, pp. 43 – 61.
78. Sullivan, G. (1994), "Logistics Management System (LMS): Integrating Decision Technologies for Dispatch Scheduling in Semiconductor Manufacturing," *Intelligent Scheduling*, Morgan Kaufman Publishers, pp. 473 – 516.
79. Sullivan G. (1995), "A Dynamically Generated Rapid Response Fast Capacity Planning Model for Semiconductor Fabrication Facilities," *The Impact of Emerging Technologies on Computer Science and Operations Research*, Kluwer Academic Publishers, Boston (presented at Winter 1994 Computers and Operations Research Conference).
80. Tayur, S., Ganeshan, R., and Magazine, M. (1998), *Quantitative Models for Supply Chain Management*, Kluwer Academic Publishers, Boston, MA.
81. Uzsoy, R., Lee, C., and Martin-Vega, L.A. (1992), "A Review of Production Planning and Scheduling Modules in the Semiconductor Industry, Part 1: System Characteristics, Performance Evaluation, and Production Planning," *IIE Transactions*, Scheduling Logistics, 24(4), pp. 47 – 60.
82. Uzsoy, R., Lee, C., and Martin-Vega, L.A. (1994), "A Review of Production Planning and Scheduling Modules in the Semiconductor Industry, Part 2: Shop Floor Control," *IIE Transactions*, Scheduling Logistics, 26(5), pp. 44 – 55.
83. Wein, L. 1988, "Scheduling Semiconductor Wafer Fabrication", *IEEE Transactions on Semiconductor Manufacturing*, Vol. 1, No. 3., 115-130.
84. Wolfson, R. (2000), "Einstein's Relativity and the Quantum Revolution," Great Courses on CD by the Teaching Company, Chantilly, VA.
85. Woolsey, G. (1979), "Ten Ways to Go Down with Your MRP," *Interfaces*, Vol. 9, No. 5, pp. 77 – 80.
86. Zisgen, H. (2005), "EPOS – Stochastic Capacity Planning for Wafer Fabrication with Continuous Fluid Models," IBM Global Engineering Services, Decision Technology Group Mainz, Germany.
87. Zisgen, H., Ments, I., Wheeler, B, and Hanschle, T., (2008), "A Queuing Network Based System to Model Capacity and Cycle Time For Semiconductor Fabrication," *Proceedings of the 2008 Winter Simulation Conference*.
88. Zisgen, H., Brown, S, Hanschke, T., Meents, I., and Wheeler, B (2010) "Queuing Model Improves IBM's Semiconductor Capacity and Lead-Time Management," *Interfaces*, Vol. 40, No. 5, pp. 397-407

Appendix 1: Basics of the Factory Operating Curve and the Trade-Off between Machine Utilization (total output) versus Lead or Cycle Time

Introduction

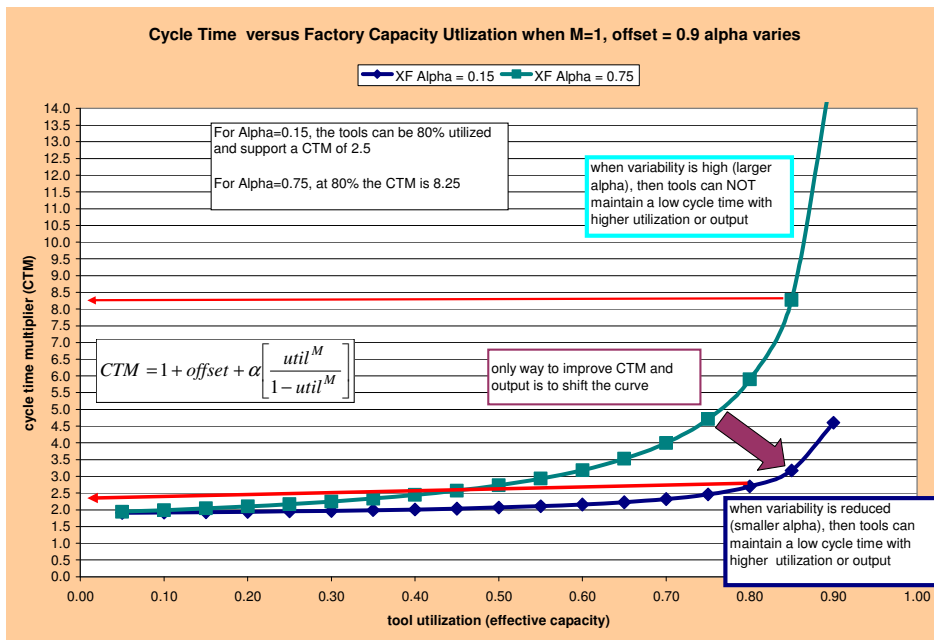
When variability exists either in arrivals or services there is a trade-off between server (machines, people) utilization and the lead or cycle time to complete an activity or service → the higher the utilization the longer the cycle time. Since the higher the server is utilized, the more total output over time, this is also referred to as the output and cycle time tradeoff. Sometimes output is referred to as the daily going rate (DGR). The curve (relationship) that describes this trade-off is called the **Operating Curve** (OC) and typically as tool utilization or output increases the cycle time increases. Typically, but not always, the relationship is a monotonically increasing non-linear function.

Cycle time is divided into two components: wait time and the actual or raw process time (RPT). Simply think about paying for your groceries at the local market. There are two components: waiting for the cashier to process your order and the actual checkout process. The time it takes to actually check out is the RPT. Often the RPT itself can be broken down into components. Again let's look at paying for groceries. There are four major activities: putting your groceries on the belt, the scanning process of each item to determine your cost (and for inventory), bagging the groceries, and paying for the groceries. Observe in some circumstances each of the four subcomponents involved in process may or may not overlap with other activities. I might put my groceries on the belt while someone else is being checked out. I might choose to bag groceries while the clerk puts them through the scanner. And interrupts can occur: the clerk runs out of singles; someone buys cigarettes which have to be fetched from a special location; bags run out; fruits, vegetables, and bakery goods in white bags have to be hand entered. For this discussion, we'll assume we can find RPT consistently and reliably.

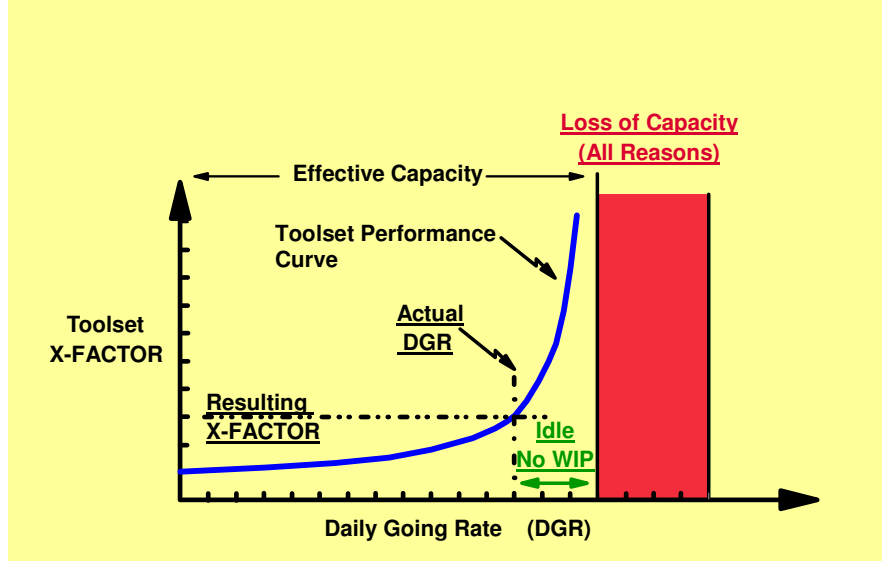
Cycle time is often measured as a cycle time multiplier (CTM), where total elapsed cycle time equals CTM times the raw process time (RPT). Sometimes the CTM is called the X factor (XF). The machine utilization is typically measured as the fraction of time the tool is busy or the average output or daily going rate (DGR) from the tool.

This relationship between cycle time and utilization or output or DGR applies to factories, banks, grocery stores, help desks, emergency rooms, etc – there is no hiding from this tradeoff. This curve is referred to as the operating curve and often looks like a hockey stick. This relationship and the curve follow naturally from queuing theory equations as well as empirical observations.

To be clear, different “players” (firms, tools, etc) have different shaped hockey sticks. The differences in shape are due to the variability in the arrivals and service times. The less variability the longer the handle and the sharper the slope when the blade curls up. For each given curve, the cost to reduce cycle time is lower tool utilization or output and the cost to increase output is higher cycle times. However, the cost (slope) changes depending on where the factory is currently operation along the curve. The left part of the curve is “flat” – low slope. Here, you can increase tool utilization (typically with more starts or load) at small additional cost in cycle time. Additionally perturbations in the work load or tool availability are easily absorbed. The right part of the curve is steep. Here any increase in starts drives a large increase in cycle time and small perturbations take a long time to work off. The only way to improve both cycle time and tool utilization at once is to take fundamental actions to “shift” the curve either by reducing variability or mining for capacity.



Line/Toolset Performance Curve

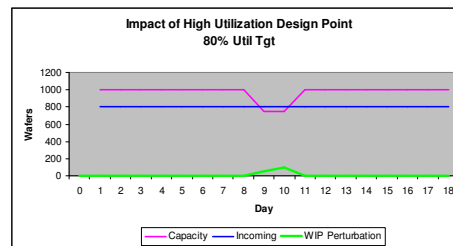
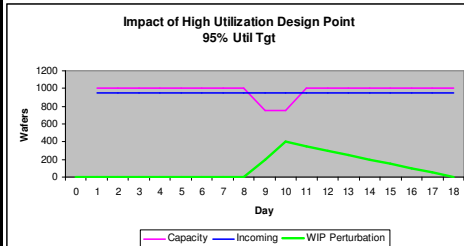


Impact of High Utilization

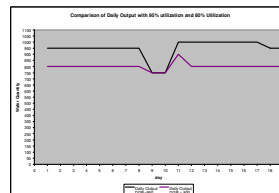
- With no variation (load or capacity) No impact. // 25% Capacity impact for 2 days:

Toolset loaded at 95%
2 Day event
7 Day recovery

Toolset loaded at 80%
2 Day event
0 Day recovery



Toolset loaded at 95%
Has higher sustained
Output then at 80%



Simplest Operating Curve

The simplest operating curve derives directly from an M/M/1 model - exponential arrival, exponential service rate, 1 machine, That derivation is:

- λ is average number of lots arriving for processing or service at the machine per unit time.
- μ is the average number of lots the machine is able to service or process per unit time.
- W is the total time the lot spends in the system (wait time and service time).
- L is the expected number of lots in the system (waiting and service)
- $L = \lambda W$ is called Little's Law which states # units in the system = arrival rate $\cdot$ total wait time. For example, if 500 units arrive a day and each unit takes 10 days to get out of the system, we have 500 in the systems. This is true independent of the distribution of the arrival and service events
- $W = L \cdot (1 / \lambda)$ Little's law solved for W
- ρ is the percentage of time the machine is busy or not idle called the utilization factor.
- $\rho = \lambda / \mu$
- $L = \lambda / (\mu - \lambda)$ -- only for M/M/1 (H&L p. 786)
- $W = 1 / (\mu - \lambda)$ -- only for M/M/1 derived from L equation and Little's law

Raw process time (RPT) is the actual time the part spends being processed by the machine. It is the service time. RPTA (RPT average) = $1 / \mu$

Cycle time (CT) is the total time it takes a part to be processed at a particular machine. CT includes the wait time and the raw process time (RPT). CTA (CT average) = W

Machine utilization (MU) is the percentage of time the machine is busy processing a part. MUA (MU average) = ρ

Cycle time is measured by a quantity called cycle time multiplier (CTM) (sometimes called XFACTOR→ XF). CTM is the ratio of average CT with average RPT.

CTM = cycle time / raw process time

CTM = CTA / RPTA = $W / (1 / \mu)$

CTM = $W / (1 / \mu) = W \cdot \mu$

– Substituting $W = 1 / (\mu - \lambda)$ we get

CTM = $\{1 / (\mu - \lambda)\} (\mu)$ – Simplifying the fraction, we get

CTM = $\mu / (\mu - \lambda)$

CTM = $\{\mu / (\mu - \lambda)\} \{(1/\mu) / (1/\mu)\}$

CTM = $(\mu / \mu) / (\mu / \mu - \lambda / \mu)$

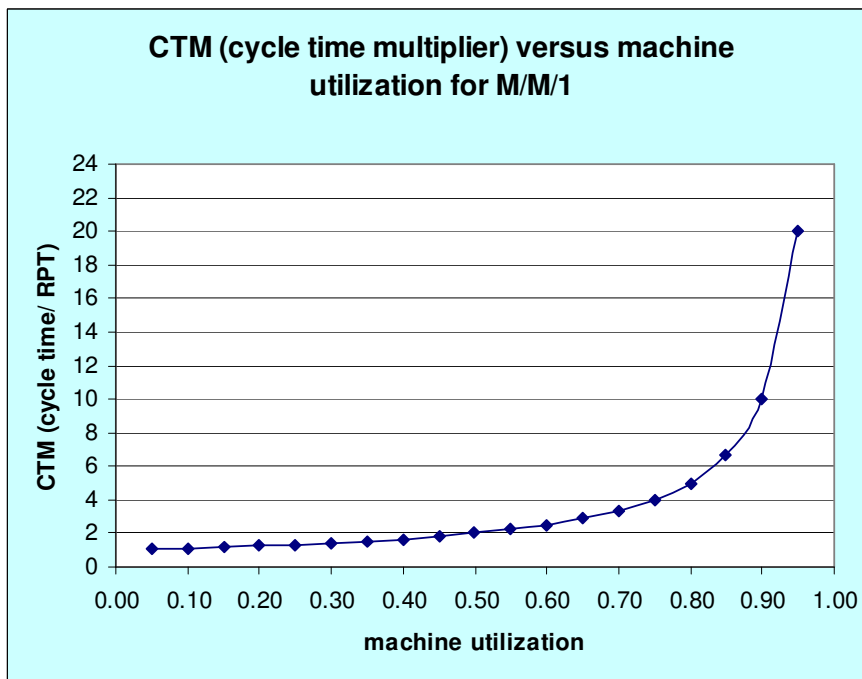
CTM = $1 / (1 - \lambda / \mu)$

CTM = $1 / (1 - \rho) = 1 / (1 - MUA)$

$$CTM = 0 + 0 + 1 \left[\frac{(1)}{1 - (util)} \right] \text{ where util = MUA = } \rho = \text{fraction of tool utilization}$$

The relation between CTM and L (number of lots in the system) for M/M/1 is provided in the last section.

As MUA increases in value, 1-MUA decreases in value, so CTM increases. Hence as we increase tool utilization, we worsen cycle time since the CTM factor increases. Below are paired values for MUA and CTM and their graph – **the operating curve**

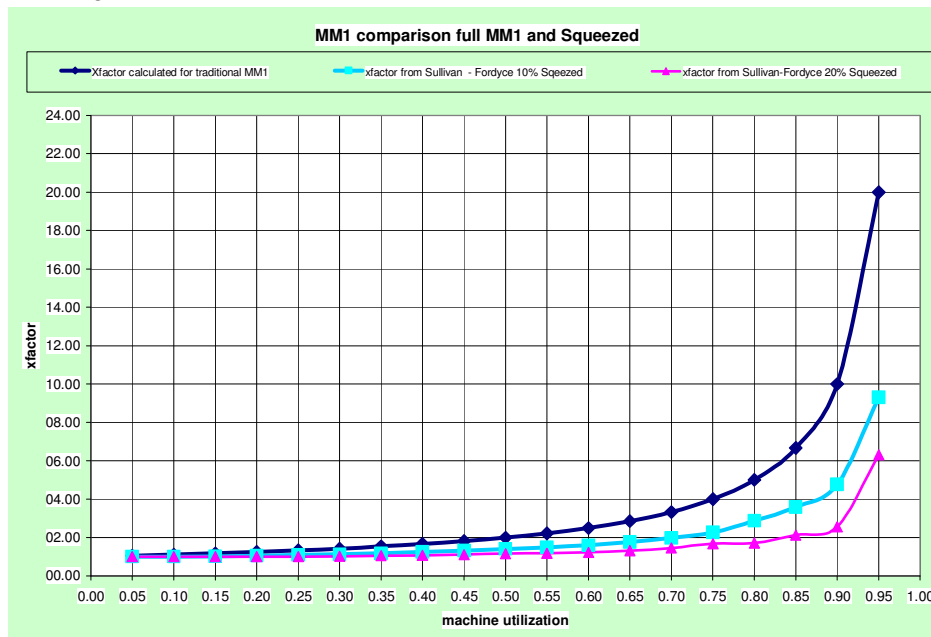


Shifting the Curve

For a given operating or trade-off curve, there is no free lunch – if you want higher machine utilization, the cost is more cycle time. However, the cost (slope) changes depending on where the factory is currently operation along the curve. The left part of the curve is “flat” – low slope. Here, you can increase tool utilization (typically with more starts or load) at small additional cost in cycle time. The right part of the curve is steep. Here any increase in starts drives a large increase in cycle time. This is called **moving along the curve** – either by increasing or decreasing starts or adding capacity.

A constant goal of factories is to take actions to *shift the curve* by reducing variability in arrivals, variability in service time, and mining for capacity. This can occur with a smarter starts pattern, reducing variability in equipment availability, improving tool fundamentals (for example faster heat up and cool down; better robot design; smarter placement of materials, etc) and smarter dispatch scheduling to better use equipment (avoid WIP piles and holes, full batches, reduced setup, etc).

The following graph shows how the curve shifts simply by squeezing the arrival rate. This reduces outliers by squeezing out very short and very long inter-arrival time and narrowing service times



A Curve that has worked well for FABs – the “DON” Curve

To refresh our memories: an operating curve is any mechanism that relates cycle time to tool utilization (or output or daily going rate) and typically looks like a “hockey stick”. That as tool utilization increases cycle time increases and there is point along the curve where the “line” takes off in a steep upward direction (the first derivative – sensitivity to change). The literature has a wealth of information on different core equations that provide “hockey stick” style curves for G/G/M models (general arrival, general service, and M servers (See H&L and Factory Physics as a starting point- also give Harris book). Fournier, Martin and Morrison (Morrison and Martin 2006 and 2007) from the IBM semiconductor facility in Burlington, VT (BTV) have identified and road tested one that often works well: This is the curve plotted in the first figure in the introduction section – we call the “DON” curve.

$$CTM = XF = 1 + offset + \alpha \left[\frac{\left(\frac{util}{util_{\max}} \right)^M}{1 - \left(\frac{util}{util_{\max}} \right)^M} \right]$$

CTM=XF is the cycle time multiplier of RPT – measure of cycle time

util – is the fraction of tool utilization in the entity – facility, tool set, checkout clerks, etc

util_{max} – is the maximum utilization for the entity, often, but not always, this value is 0.95 or 1.0

1 – since CTM is a multiple of RPT, this sets the minimum as 1 when util and offset are 0.

offset – are various of aspects of the process that generate wait time and really can not be eliminated. For example: travel time, hold hours, post processing hours relative to total RPT. A typical value for offset is 0.9. When offset is 0.9 this sets the minimum CTM at 1.9

M is the number of machines or servers. Typically this value ranges from 1 to 4. Fournier, Morrison, and Martin report values greater than 3 or 4 (even when the number of tools exceeds 4) work best.

α represents the amount of variation in the system and controls how long the curve stays flat. The lower the **α** the less variation and longer the curve stays flat. Typical values for α range

$$\alpha = \frac{\sqrt{C_a^2 + C_s^2}}{2}$$

between 0.35 and 0.65, but not always. Formally where C_a is the coefficient of variability in lot arrivals at each oper and C_s is the coefficient of variability in lot service through every operation.

Often in tool planning, we wish to specify a CTM value and calculate the utilization required to support this cycle. When we solve the previous equation we get

$$util = util_{\max} \left(\frac{CTM - (1 + offset)}{CTM - (1 + offset) + \alpha} \right)^{1/M}$$

For tool planning, often the version with util_{max} = 1 is used and the equations become

$$CTM = 1 + offset + \alpha \left[\frac{util^M}{1 - util^M} \right]$$

$$util = \left(\frac{CTM - (1 + offset)}{CTM - (1 + offset) + \alpha} \right)^{1/M}$$

When **M=1**, **util_{max}=1**, and **offset = 0.9** the Equation for CTM is

$$CTM = XF = 1 + 0.9 + \alpha \left[\frac{(util)}{1 - (util)} \right]$$

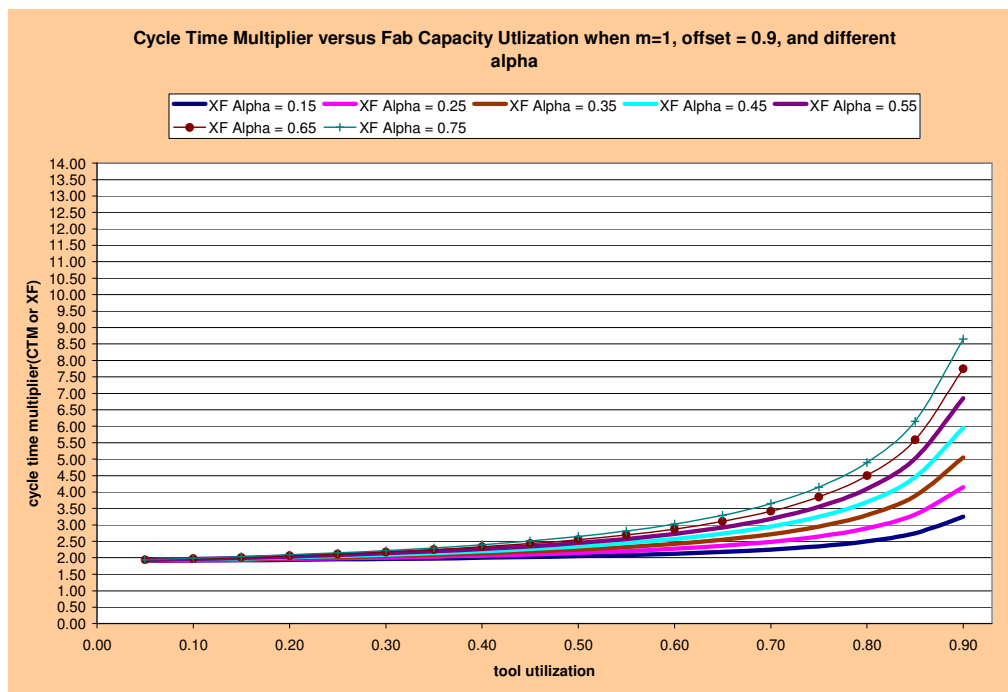
$$= 1.9 + (\alpha)(util) \left[\frac{1}{1 - (util)} \right]$$

Observe the similarity to the equation for CTM in the M/M/1 model

$$CTM = 0 + (1)(1) \left[\frac{(1)}{1 - (util)} \right]$$

M/M/1 has no offset and its alpha is 1 (high variation)

This graph shows a series of curves for different alphas when offset = 0.9. M=1, $util_{max} = 1$



PK Mean Value Formula for M/G/1

Another equation to relate cycle time and tool utilization is the PK (Pollaczek-Khinchin) mean value formula for M/G/1 (exponential arrivals, general service, 1 server).

Consider an M/G/1 queue

- Time between arriving lots Poisson process rate λ (M)
- Service time - general distribution with coefficient of variation c_s (G), and rate μ

Pollaczek-Khinchin mean value formula

- CTM = XF = Cycle time/process time = normalized cycle time
- $CTM = XF = 1 + (1/2) * (1 + c_s^2) * (\lambda/\mu) / [1 - (\lambda/\mu)]$
 $= 1 + (1/2) * (1 + c_s^2) * (UTIL) / (1 - UTIL)$
 $= 1 + \alpha * (UTIL) / (1 - UTIL)$

$$XF = 1 + \alpha \frac{\rho}{1 - \rho}$$

$$XF = \frac{1 - \rho}{1 - \rho} + \alpha \frac{\rho}{1 - \rho}$$

$$XF = \frac{1 - \rho}{1 - \rho} + \frac{\alpha \rho}{1 - \rho}$$

$$XF = \frac{(1 - \rho) + \alpha \rho}{1 - \rho}$$

$$XF = \frac{1 - \rho + \alpha \rho}{1 - \rho}$$

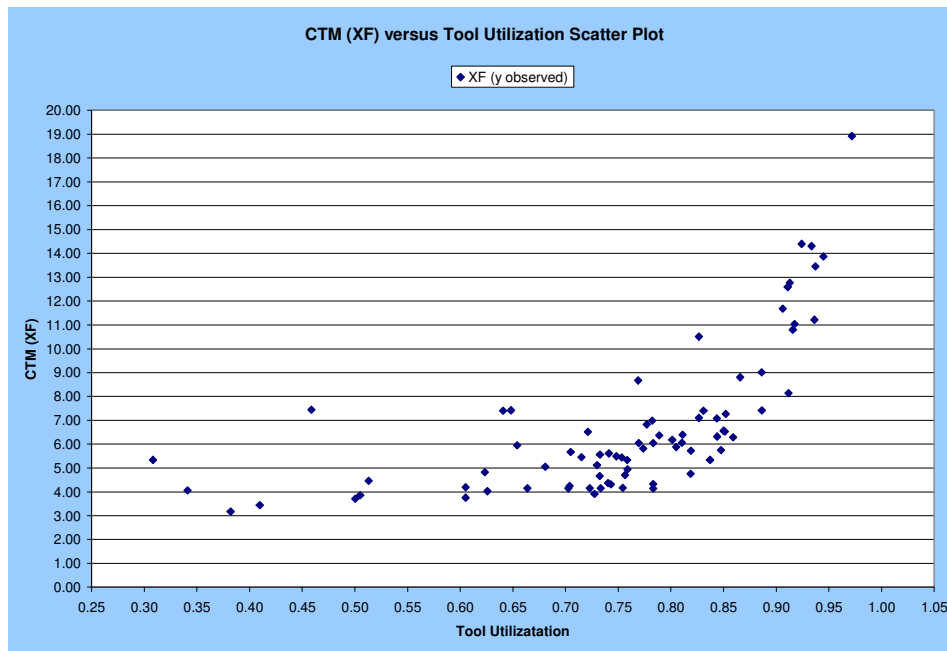
$$XF = \frac{1 + (\alpha - 1)\rho}{1 - \rho}$$

Working with Observed data

Typically a factory keeps a large quantity of information for each key tool set including and operations history which has such information as when a lot arrived at the tool set, which tool handled this lot, when the lot was loaded on the tool, when the manufacturing process began, when the manufacturing process ended, when the lot was unloaded from the tool, and when the lot left this tool area. Often, it saves multiple years worth of data summarized by week which has the tool utilization for the week and the CTM or XF for this week. Below is sample of this type of data.

numeric	charac	observed Values over time	
Week	xlabel	utilization (x value)	XF (y observed)
200901	2009 01	0.65	7.42
200902	2009 02	0.34	4.06
200903	2009 03	0.31	5.34
200904	2009 04	0.38	3.17
200905	2009 05	0.41	3.45
200906	2009 06	0.46	7.44
200907	2009 07	0.51	4.47
200908	2009 08	0.51	3.85
200909	2009 09	0.50	3.71
200910	2009 10	0.61	3.75
200911	2009 11	0.63	4.02
200912	2009 12	0.62	4.83
200913	2009 13	0.60	4.19
200914	2009 14	0.66	4.14
200915	2009 15	0.68	5.05
200916	2009 16	0.74	5.61
200917	2009 17	0.72	6.51
200918	2009 18	0.78	6.99
200919	2009 19	0.70	5.67
200920	2009 20	0.73	4.16
200921	2009 21	0.76	4.94
200922	2009 22	0.73	5.12
200923	2009 23	0.76	5.33
200924	2009 24	0.78	4.32
200925	2009 25	0.78	4.14

A typical first step is to create a scatter plot of the data

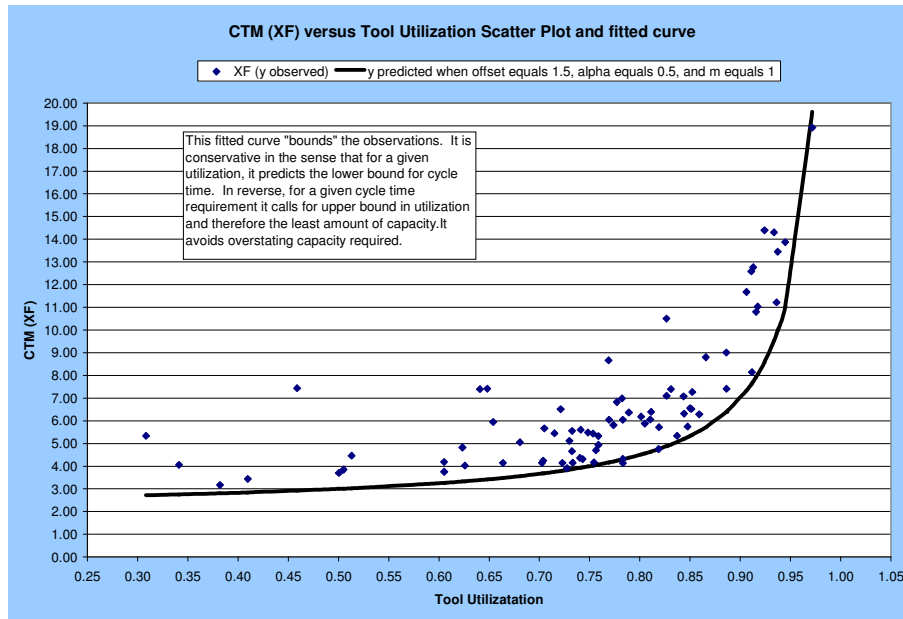


What now?

Typically what we need for planning are two items

1. a visualization of the nature of trade-off between utilization and cycle time for this entity (factory, tool set, etc) – when does the steep ascent start; how steep is the ascent; what is the variation the entity sees in cycle time for a given utilization; or the reverse
2. a method to calculate the utilization required to meet a specific cycle time target – both a single value and probability interval

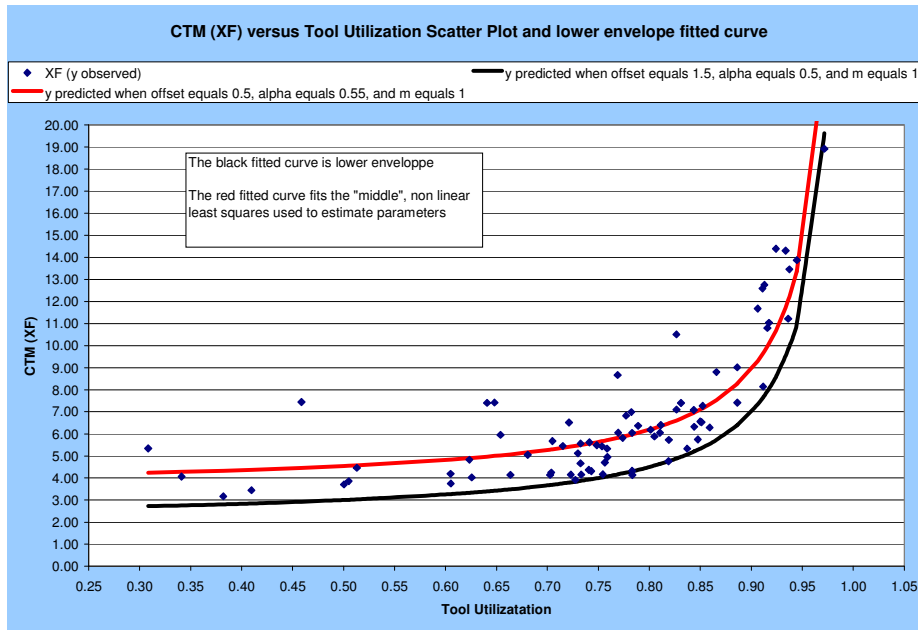
A first step (and sometimes the only necessary step) is to manually fit a curve to the data points that mirrors the shape of the data and traces or bounds the lower envelope of the data. This can be done quickly with trial and error. For this data the manually fitted curve has offset=1.5, M=1, and $\alpha = 0.5$. The graph below shows this fitted curve with the data points.



This fitted curve "bounds" the observations. It is conservative in the sense that for a given utilization, it predicts the lower bound for cycle time. In reverse, for a given cycle time requirement it calls for upper bound in utilization and therefore the minimum capacity needed to meet business objectives.

There are some formal mathematical methods to automatically find a curve that meets these criteria; but they require some non-standard statistical methods.

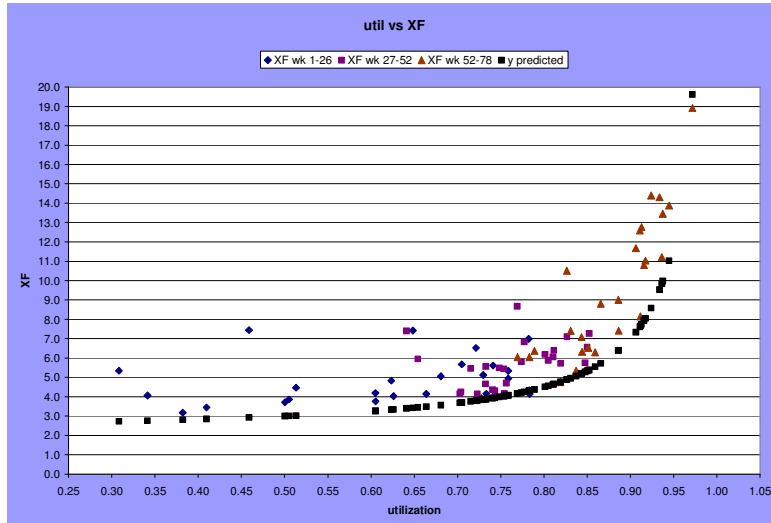
A second option is to find a curve that hits the middle of the population. "Middle" can be defined as finding offset and alpha finds minimizes the deviation between the fitted data points and the curve. This statistical method is called non-linear least squares. The graph below has this fit.



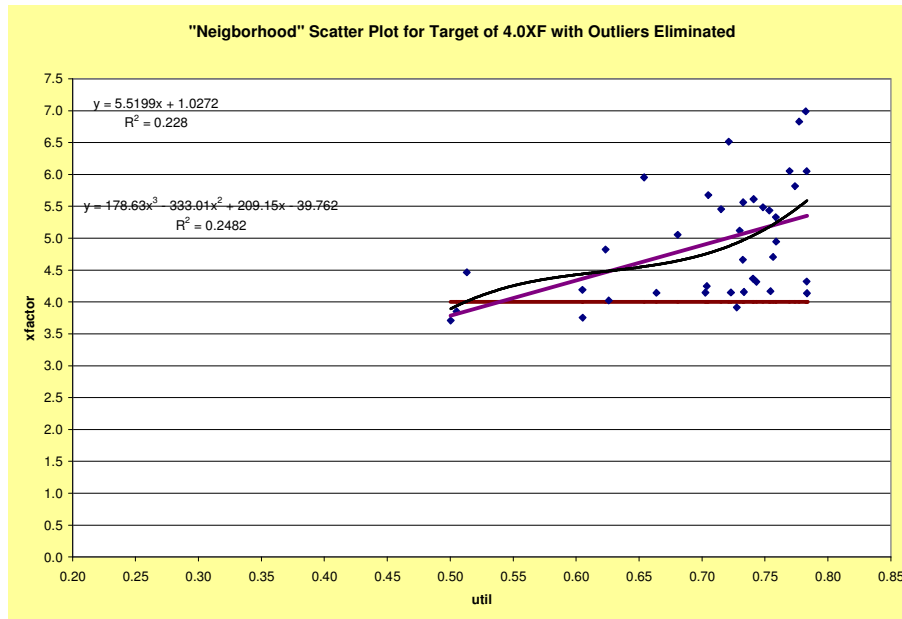
Although both approaches do a reasonable job in meeting goal 1 (visual), they fail to address some requirements for goal 2 (estimating utilization needed to achieve cycle time goal) such as:

- prediction and confidence intervals
- time series nature of the data
- changes in core structures over time

For example is there time series pattern in the data? The graph below is a scatter plot of the same data points where we can identify which time group a point belongs to.



A third method is called “neighborhood” analysis and involves a computational based method that looks at points local to the cycle time of interest. Graph below provides an example for CTM=4.0



Linking effective capacity to the cycle time commit in Capacity Planning?

How do we mathematically adjust the effective capacity available to met demand volume based on demand lead (cycle time) commitment made – or the reverse? We create an “up lift factor” - ULF

Let’s start by reviewing what the Operating Curve (OC) doe for us?

It enables to gauge the trade-off between tool utilization or effective capacity and cycle time (given as a cycle time multiplier (CTM) or X Factor (XF) = cycle time / raw process time). If we are given utilization, we can estimate an XF with this equation:

$$CTM = XF = 1 + offset + \alpha \left[\frac{util^M}{1 - util^M} \right]$$

If we are given a target CTM, we can estimate the utilization needed to achieve that CTM

solving the previous equation for util: $Util = \left(\frac{CTM - (offset + 1)}{CTM - (offset + 1) + \alpha} \right)^{\frac{1}{M}}$

How doe we use this in tool or capacity planning to create the Up Lift Factor?

Assume that the product XYZ is processed 5 times by tool set AAA during its production or route. Each time a wafer goes through tool set AAA it is referred to as a “pass”. In this case product XYZ has 5 passes on tool set AAA. Additionally, assume the average RPT for each XYZ wafer on tool set AAA is 2 units. In steady state, this makes for a total RPT required per day of 10 (=5x2) units per wafer of XYZ on tool set AAA.

Assume we start 20 wafers per day, then the RPT required is 200= (20x10) units. However, if we had just enough capacity to handle the 20 wafers (200 units), then the utilization would be 100%, so the cycle time would be infinite (if there is any variation in arrivals or service)

Case 1- Tool Set Utilization is specified

What if the factory was willing to spend enough money to buy extra tooling so the toolset could operate at 80% utilization (20% of the time the tool was idle without work) – how much tool time would we need? The answer is $250 = 200 / 0.80$ units. If have 250 units of RPT and we use 200 units of RPT for wafers, then we have a spare 50 units -- $200/250 = 0.80 = 80\%$ utilization.

Observe we can rewrite the equation $250=200 / 0.80$ as $250 = 200 \times (1/0.80) = 200 \times 1.25$. The value 1.25 is called the uplift factor (ULF - referred to as the Wafer Term Multiplier – WTM).

$$ULF = WTM = \frac{1}{\text{toolset\_utilization}} = \frac{1}{ts\_util}$$

Formally, we can define ULF as follows. Given a workload on a tool set specified as RPT time units called REQ_RPT (required RPT for this tool set for this product) and a utilization percentage specified for this tool set call TS_UTIL (tool set utilization), then ULF equals $1/TS\_UTIL$ and is the factor to multiply REQ_RPT to determine the total capacity (in time units) required (CAP_REQ) for this tool set to meet the workload and maintain desired tool utilization levels

WSD = wafer starts per day for this product = 20

PASS_COUNT = number of passes this product makes on this toolset = 5

RPT_AVE = average RPT per pass per wafer = 2

REQ_RPT = total RPT required per day for this product for this wafer load for this toolset

REQ_RPT = WSD x PASS_COUNT x RPT_AVE = $20 \times 5 \times 2 = 200$

TS_UTIL = the utilization level = 0.80

ULF = up lift factor = $1 / TS\_UTIL = 1 / 0.80 = 1.25$

CTM = XF = cycle time this tool set can support with specified level of utilization = 4.0
= first equation above

CAP_REQ = capacity required to produce wafers and meet cycle time
= REQ_RPT x ULF = $200 \times 1.25 = 250$

Case 2- Cycle time (CTM) is specified

The business states it wants to run this product with a CTM of 4.0. This requires some portion of time the tool set is available to produce, but does not have WIP. How do we incorporate that into the planning process?

We calculate a burden or uplift factor (ULF) per widget based on the target CTM and the specific characteristics of the Operating Curve for this tool set. Assume the Operating Curve for this tool set has offset=1, alpha=0.5, and m=1. Then using the equation solved for UTIL, that the required utilization to achieve the CTM target of 4.0 is 0.80 (80%). For each unit of raw capacity required, we need 1.25 units available to meet the CTM target. The 1.25 is the uplift factor (ULF)

$$ULF = 1 / \text{tool\_utilization\_meet\_cycle\_time\_target\_from\_opcurve}$$

If we have 250 units of capacity available per day, how many widgets can we start per day at committed cycle time? The answer is $20 = \{250 / (1.25 \times 10)\} = 250 / 12.5$.

If the business wanted to achieve a cycle time of 3.5, how many widgets could it start per day? Using the same equation for UTIL, the utilization required to achieve a cycle time of 3.5 is 0.75 (75%). The ULF is $1.333 = 1 / 0.75$. The maximum number widgets per day at this cycle time commit is $\{250 / (1.333 \times 10)\} = 18.75$. Shorter cycle time, reduced widget starts.

General Approach modify the capacity required

Alternatively, instead of decreasing the capacity available, we can up lift the capacity required per unit of production. For the 4.0 cycle time commit, we up lift 10 units per day per widget to 12.5 (10 x 1.25) For the 3.5, the 10 units is uplifted to 13.33 (10 x 1.33).

The set of equations are:

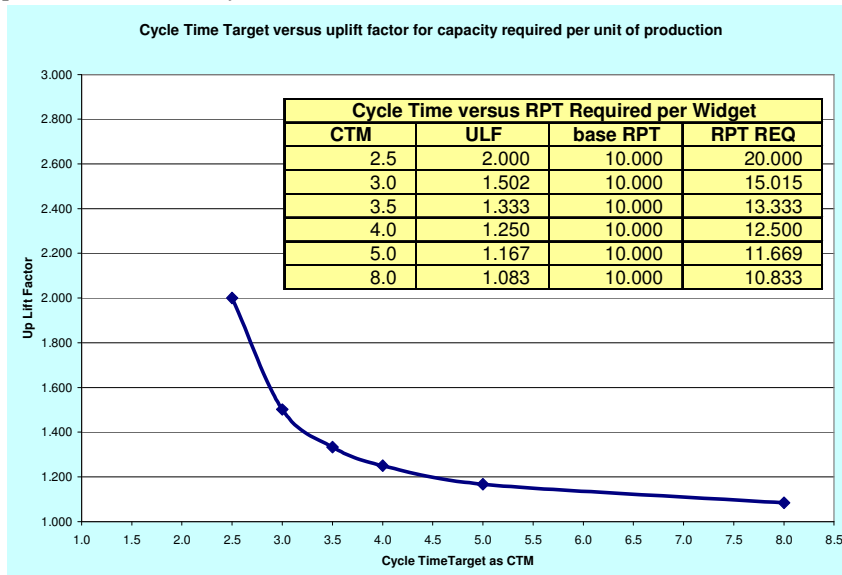
$$1. \quad REQ\_RPT = WSD \times PASS\_COUNT \times RPT\_AVE - \text{minimum required raw process for each unit of production}$$

$$2. \quad Util\_REQ = \left(\frac{CTM\_REQ - (offset + 1)}{CTM\_REQ - (offset + 1) + \alpha} \right)^{\frac{1}{M}} \quad \text{the utilization required to achieve the cycle time required is calculated with this equation}$$

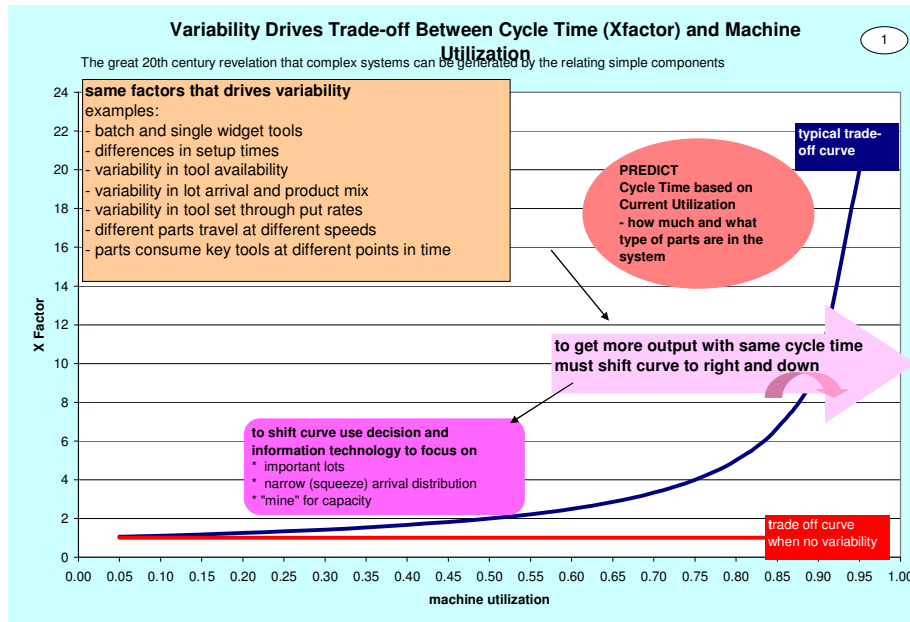
$$3. \quad ULF = WTM = \frac{1}{Util\_REQ} \quad \text{the uplift factor is the reciprocal of the utilization}$$

$$4. \quad CAP\_REQ = REQ\_RPT \times ULF \quad \text{this the capacity required to produce 1 unit of this part at the specified cycle time given the number of passes on the tool and average RPT.}$$

The following figure gives us the uplifted capacity required to input to the central planning process based on the cycle time the business wishes to achieve.



Summary – Variability Drives the Trade-off Between Cycle time and Machine Utilization



Bibliography

- Bitran, G. and Tirupati, D. (1989), "Tradeoff Curves, Targeting and Balancing in Manufacturing Networks", *Operations Research*, Vol. 37, No. 4, pp. 547- 555
- Chen, H., Harrison, M., Mandelbaum, A., Ackere, A., and Wein, L. (1988), "Empirical Evaluation of a Queuing Network Model for Semiconductor Wafer Fabrication", *Operations Research*, Vol. 36, No. 2, pp. 202-215.
- Hopp, W. and Spearman, M. (2008), *Factory Physics, 3rd Edition*, McGraw-Hill Irwin, NY
- Miller, D. 1990, "Simulation of a Semiconductor Manufacturing Line", *Communications of the ACM*, Vol. 33, No. 10, pp. 98-108.
- Morrison, J. and Martin D, 2006, "Cycle Time Approximations for the G/G/m Queue Subject to Server Failures and Cycle Time Offsets with Applications", ASMC.
- Morrison, J. R. and D. P. Martin. 2007. Practical extensions to cycle time approximations for the G/G/m-queue with applications. *IEEE Transactions on Automation Science and Engineering*. 4 (4): 523-532.
- Zisgen, H. et al (2010), "Queueing Model Improves IBM's Semiconductor Capacity and Lead-Time Management", *Interfaces*.

Relation between CTM and L (number of lots in the system) for M/M/1

We can relate CTM back to L (number of lots in the system – waiting and on the tool) as follows

$$L = \lambda / (\mu - \lambda)$$

$$L = \{\lambda / (\mu - \lambda)\} \cdot \{(1/\mu) / (1/\mu)\}$$

$$L = (\lambda/\mu) / ((\mu/\mu) - (\lambda/\mu))$$

$$(\lambda/\mu) = \rho = \text{MUA}$$

$$(\mu/\mu) = 1$$

$$L = \text{MUA} / (1 - \text{MUA})$$

$$L = 1 / (-1 + (1/\text{MUA}))$$

$$\text{CTM} = 1 / (1 - \text{MUA}) \leftrightarrow \text{MUA} = 1 - (1/\text{CTM})$$

when we put the equation for MUA in terms of CTM into the equation for L we get

$$L = (1 - (1/\text{CTM})) / (1 - (1 - (1/\text{CTM})))$$

$$L = (1 - (1/CTM)) / (1/CTM)$$

$$L = \{(1 - (1/CTM)) / (1/CTM)\} \cdot \{CTM/CTM\} = (CTM - 1) / 1 = CTM - 1$$

$$L = CTM - 1$$

Some sample data is

MUA	CTM	L
0.05	1.0526	0.0526
0.10	1.1111	0.1111
0.15	1.1765	0.1765
0.20	1.2500	0.2500
0.25	1.3333	0.3333
0.30	1.4286	0.4286
0.35	1.5385	0.5385
0.40	1.6667	0.6667
0.45	1.8182	0.8182
0.50	2.0000	1.0000
0.55	2.2222	1.2222
0.60	2.5000	1.5000
0.65	2.8571	1.8571
0.70	3.3333	2.3333
0.75	4.0000	3.0000
0.80	5.0000	4.0000
0.85	6.6667	5.6667
0.90	10.0000	9.0000
0.95	20.0000	19.0000
0.99	100.0000	99.0000

Algebra to solve CTM equation for UTIL

We can solve for util as follows

$$CTM = 1 + offset + \alpha \left[\frac{\left(util / util_{\max} \right)^M}{1 - \left(util / util_{\max} \right)^M} \right]$$

let $X = util / util_{\max}$ and $A = 1 + offset$ then

$$CTM = A + \alpha \left[\frac{X^M}{1 - X^M} \right]$$

$$\frac{CTM - A}{\alpha} = \frac{X^M}{1 - X^M}$$

$$(CTM - A)(1 - X^M) = \alpha X^M$$

$$(CTM - A) - (CTM - A)X^M = \alpha X^M$$

$$(CTM - A) = \alpha X^M + (CTM - A)X^M$$

$$(CTM - A) = (\alpha + CTM - A)X^M$$

$$\frac{(CTM - A)}{(\alpha + CTM - A)} = X^M$$

$$X = \left(\frac{CTM - A}{\alpha + CTM - A} \right)^{1/M}$$

$$X = \left(\frac{CTM - A}{CTM - A + \alpha} \right)^{1/M}$$

Making the reverse substitutions we get

$$\frac{util}{util_{\max}} = \left(\frac{CTM - (1 + offset)}{CTM - (1 + offset) + \alpha} \right)^{1/M}$$

$$util = util_{\max} \left(\frac{CTM - (1 + offset)}{CTM - (1 + offset) + \alpha} \right)^{1/M}$$

Stochastic scheduling: A short history of index policies and some key recent developments

K.D. Glazebrook · D.J. Hodge
C. Kirkbride · R.J. Minty

Abstract A *multi-armed bandit problem* classically concerns $N \geq 2$ populations of rewards whose statistical properties are unknown (or at least only partly known). A decision-maker secures rewards by sampling sequentially from the populations, using past sampled values to make inferences about the populations and so guide the choice of which population to sample next. The goal is to make these choices in such a way as to maximise some measure of total reward secured. Such problems embody, in a particularly simple form, the dichotomy present in many decision problems between making decisions with a view to securing information which can improve future decision-making (*exploration*) and those which exploit the information already available (*exploitation*). In the 1970's John Gittins discovered that important classes of such multi-armed bandit problems have solutions of a particularly simple form: at each stage of the sampling compute an *index* (the *Gittins index*) for each of the N populations, namely a function of the rewards already sampled from the population concerned. Always sample next from the population with the largest current index. Moreover, the index concerned has a simple interpretation as an equivalent known reward for the population concerned. It emerges that many problems involving the sequential allocation of effort, some of quite different character to the above multi-armed bandit problems, have index solutions. Since the 1970's, Gittins' index result together with a range of developments and reformulations of it have constituted an influential stream of ideas and results contributing to research into the scheduling of stochastic objects. Application areas to which these ideas have contributed include approximate dynamic programming, con-

K.D. Glazebrook
Department of Management Science, Lancaster University, LA1 4YX, UK.
E-mail: k.glazebrook@lancaster.ac.uk

D.J. Hodge
School of Mathematical Sciences, University of Nottingham, Nottingham, NG7 2RD, UK.
E-mail: david.hodge@nottingham.ac.uk

C. Kirkbride
Department of Management Science, Lancaster University, LA1 4YX, UK.
E-mail: c.kirkbride@lancaster.ac.uk

R.J. Minty
School of Mathematics, Cardiff University, Cardiff, CF24 4AG, UK.
E-mail: MintyRJ@cardiff.ac.uk

trol of queueing systems, fast fashion retail, machine maintenance, military logistics, optimal search, research planning, sensor management, communication channel usage and website morphing. The talk will give an overview of some key ideas and some recent developments.

1 Introduction

The term *multi-armed bandit* is sometimes reserved for a class of problems in which the goal is to sample sequentially from a collection of reward populations whose statistical properties are not fully known in order to maximise some measure of the total reward sampled. In fact the critical contribution to such problems made by Gittins [10] had its roots in new-product pharmaceutical research where the challenge was to conduct a search among families of compounds in order to find one whose therapeutic effect exceeded some target and to do so as quickly as possible. The nature of the solution to the problems studied by Gittins was that to each candidate reward population or family of compounds could be attached an *index*, a function of the outcomes for that candidate to date, such that at each stage the candidate to be chosen next was optimally the one of highest index. Such *index policy* solutions have both intuitive power and computational advantage. It soon became clear that the optimality of such index policies could be established for a wider class of models for *dynamic resource allocation* in which some *single indivisible resource* was available for allocation to a collection of reward generating/cost incurring stochastic projects (sometimes called *bandits*) which are in competition for it. Crucially, in Gittins' models all projects *not* in receipt of resource must remain fixed and projects must evolve independently under the allocation of resource. We shall use the term *multi-armed bandits* (MABs) for this wider class of models which will be described in more detail in Section 2. Whittle [36] expanded considerably the scope of index ideas by developing an index heuristic for a class of *restless bandits* (RBs) which generalised the Gittins models by allowing evolution of projects *not* in receipt of the resource. The last thirty years has seen a range of new insights around and extensions to *index theory*, many of which are described in Gittins, Glazebrook and Weber [11]. We give an abbreviated account of some of the key developments in Section 2.

There is little doubt that the above classical models and associated theory are compelling when applicable. Indices offer interpretable measures of the cost effectiveness of competing decision options which are usually easy to compute. They often give a route to understanding how assumed values of model parameters impact on (good) decisions. The associated index policies can be readily understood and often perform strongly. However, index theory has hitherto been limited to models for dynamic resource allocation which take a *very* simple view of the resource to be allocated. The classical models and associated extensions mostly concern *single server* problems in which the available resource is allocated *en bloc* to a single project at each decision epoch while some have contemplated extensions to configurations involving *parallel servers*. In light of such limitations, Glazebrook, Hodge and Kirkbride [14], GHK in the following, have proposed a radical extension of index theory to a class of models for dynamic resource allocation in which some fixed amount of a *divisible resource* is available for distribution among a collection of stochastic projects and which may be applied at a *range of levels* to each. The MABs of Gittins [10] and the RBs of Whittle [36] may be recovered as special cases. This model class is described in Section 3 as is the analysis based

on a Lagrangian relaxation (L) of the problem which yields the *project indices*. These indices depend *not only* upon a given project state (as in existing theory) but *also* upon a given level of resource. An index may be understood as a *fair charge* for raising the rate at which the resource is applied to the project above the given level when it is in the given state. A natural index policy will, in any system state, use these fair charges greedily to determine the distribution of the resource among the constituent projects. Such an approach will prefer an additional commitment of resource to one project rather than another if the corresponding fair charge (index) is higher.

The ideas sketched in the previous paragraph and described in Section 3 have the potential to broaden very considerably the range of problems to which index theory can be applied. However, there is a major issue which could inhibit progress. The development of GHK's project indices is contingent upon each project exhibiting a structural property which they call *full indexability*. This property is what is required to guarantee the existence of (optimal) solutions to the Lagrangian relaxation L which have index form. Notwithstanding the success of GHK in demonstrating full indexability for a small number of models for which project evolution is of birth-death type, previous experience of the much simpler class of RBs suggests that this will rarely be possible. This paper responds to this challenge in Sections 4-6 by developing an approximative index theory and by reporting an empirical investigation of its effectiveness in the context of two model classes of independent interest. The crux of this approximative theory is the suggestion that we should seek strong (though usually suboptimal) solutions to Lagrangian relaxation L of index form. To this end, we deploy dynamic programming (DP) policy improvement. We then construct *greedy index heuristics* using these approximating indices and investigate their performance computationally.

In Section 4 we develop our approximative index theory by introducing the notion of an *index generating family of policies* and the associated indices. In a sense explained in the text, index-generating families can be thought of as *laws of diminishing returns* for the constituent projects. In Sections 5 and 6 we demonstrate the power of these ideas in the context of two important model classes. In neither model class is project evolution of birth-death type. In Section 5 we present an extension to a queueing control problem discussed in GHK in which a team of servers is available to provide service at a collection of stations to which customers arrive in batches. The allocation of the team dynamically responds to the evolving queue lengths at the stations to minimise a measure of overall waiting time for the customers. In Section 6 we consider a general model class in which assets are subject to deterioration and consequentially escalating costs under use. This deterioration may be arrested by application of service resource which can return an asset to a pristine state. Here the allocation of service resource responds dynamically to the deteriorating states of the individual assets. The application cited is to machine maintenance but the model may apply in other contexts involving the management of multiple assets. In both Sections 5 and 6 the theoretical discussion of the ideas underlying the index heuristics proposed is accompanied by supporting numerical evidence of their strength of performance.

2 A short history of index policies

We use the notation $\{B_k, 1 \leq k \leq K\}$ to denote a *multi-armed bandit* (MAB) in which at each time $t \in \mathbb{N}$ a single resource is allocated to one of K stochastic projects (or *bandits*). Each bandit B_k is characterised by its *state space* Ω_k which we shall assume

to be finite or countable, a Markovian *transition law* P_k which determines how the state of B_k changes when the resource is allocated to it (i.e., when it is *continued* or *processed*), a positive-valued *reward function* $r_k : \Omega_k \rightarrow \mathbb{R}^+$ and a *discount factor* $a \in (0, 1)$. Should B_k be chosen for processing at some $t \in \mathbb{N}$ when in state $x_k \in \Omega_k$, the following ensue: a discounted reward $a^t r_k(x_k)$ is earned and the new state of B_k at time $t + 1$ is determined according to the law $P_k(\cdot | x_k)$. All project states are available to the decision-maker at all epochs $t \in \mathbb{N}$. The goal is to determine a *policy*, a rule for choosing which project to process in all possible system states, in order to maximise the total expected reward earned over an infinite horizon. While it is possible to frame this discussion in terms of more general models, the MAB $\{B_k, 1 \leq k \leq K\}$ has all of the key ingredients for our discussion. Easily secured extensions of the theory are to problems involving cost minimisation rather than reward maximisation and to those which consider a (weaker) time average reward criterion rather than one based on discounting.

In principle, it is possible to determine optimal policies for the MAB $\{B_k, 1 \leq k \leq K\}$ by use of the methods of dynamic programming (DP). In practice the size of the system state space $\mathbf{X}_k \Omega_k$ will render this computationally infeasible for problems of realistic size. It was Gittins and Jones [12] who first demonstrated that optimal policies have an *index structure*. This means that there exist *index functions* $\nu_k : \Omega_k \rightarrow \mathbb{R}$, $1 \leq k \leq K$, such that in system state $\mathbf{x} = \{x_k, 1 \leq k \leq K\}$ the choice of any project k^* for processing which satisfies

$$\nu_{k^*}(x_{k^*}) = \max_{1 \leq k \leq K} \nu_k(x_k) \quad (1)$$

is optimal. We denote such a policy by $\mathbf{u}_\nu$. Hence the problem of determining an optimal policy is reduced to the problem of determining the *index functions* ν_k .

Gittins and Jones based their proof of this index result on an intricate argument involving pairwise interchanges after the fashion used frequently in (deterministic) scheduling theory. Many further proofs of the result have followed as researchers have sought to understand why the result holds and how best to understand and exploit it. Whittle [34] gave a proof based on the theory of DP in which he demonstrated that the value function $V^{\mathbf{u}_\nu}(\mathbf{x})$, namely the total expected reward earned by operating policy $\mathbf{u}_\nu$ from initial state $\mathbf{x}$, satisfies the DP optimality equations. Weber [31] gave an extraordinarily simple and beautiful proof, involving almost no explicit mathematical formalism, based on the notions of a *prevailing charge* and a *fair charge* for processing a bandit. In that discussion, the index $\nu_k(x_k)$ is a fair charge for processing bandit B_k when in state x_k . Tsitsiklis [28] gave a proof for MABs with finite state spaces which combined a simple interchange argument with a mathematical induction on the size of the state space. Finally, the work of Tsoucas [29] and Bertsimas and Niño-Mora [3] produced a radically new perspective on index ideas and gave accounts which used the so-called *achievable region approach* to produce proofs. These used linear programming (LP) arguments applied to a reformulation of the problem as an LP whose feasible space is the set of system performances (suitably defined) achievable under some policy for the MAB. We shall later describe some important further developments of index theory which spun off from these latter contributions. Gittins' index has been utilised in a myriad of application areas, some of which are mentioned in the abstract. Others are described in Gittins, Glazebrook and Weber [11].

Whittle [34] proposed the following characterisation of a class of index functions which determine optimal policies for MABs as in (1) and coined the term *Gittins in-*

dices. Consider the decision process $\{B_k, m\}$ in which the bandit B_k is enhanced by the addition of *retirement option* m . In this new process, at each time $t \in \mathbb{N}$, the decision-maker must choose between processing B_k , with all of the attendant consequences described above, and the retirement option m in which case a fixed reward equal to $a^t m(1 - a)$ is earned and there is no change of state. It is plain that once the decision-maker optimally chooses the retirement option she will continue to do so. Hence if m is chosen optimally for the first time at t , the decision-maker effectively mandates retirement (i.e., from any further processing of the bandit) at t with a one-off *retirement reward* equal to $a^t m$. When the bandit B_k is in state x_k the Gittins index $\nu_k(x_k)$ is that unique value of m for which *both* available actions (processing the bandit and retirement) are optimal. Katehakis and Veinott [21] gave an alternative characterisation of the index $\nu_k(x_k)$ by considering bandit B_k , deemed to be in state x_k at time 0, and developing from it a reward discounted Markov Decision Process. This derived process is similar to $\{B_k, m\}$ except that, instead of offering retirement as an alternative to the conventional processing of B_k at each decision epoch, we offer instead an instantaneous return to the initial state x_k . It then emerges that $\nu_k(x_k)$ is the total expected discounted reward earned over an infinite horizon under an optimal policy for this so-called restart-in- x_k problem. With this characterisation, DP value iteration is then available for the computation of the index. Effective approaches to the computation of the index function $\nu_k(\cdot)$ have been proposed which compute index values in order from largest to smallest. The earliest proposals in this vein were due to Robinson [26] and Varaiya, Walrand and Buyukkoc [30]. A recent variant on this theme is due to Sonin [27]. Niño-Mora [24] has proposed an algorithm for index computation which has running time $\frac{2}{3} |\Omega_k|^3 + O(|\Omega_k|^2)$.

Glazebrook [13] considered a model in which each bandit B_k was like a *job* in that it was *completed* when its state entered some *completion* subset of Ω_k . He further supposed that the policies for the collection $\{B_k, 1 \leq k \leq K\}$ were delimited by imposition of *precedence constraints* whose digraph representation was in the form of an out-forest. Hence in this model each job-like bandit awaited the completion of at most a single *predecessor* in order for its processing to begin. Glazebrook established an index result for this model which essentially gave to each sub-collection of jobs within a single connected component of the digraph a Gittins index. This was enough, since at most one job from each sub-collection was available for processing at any epoch. This model captured a particularly simple form of *arrivals* pattern for bandits in that, in the model, once a job-like bandit was completed all of its *successor* jobs were then available for processing, i.e. had arrived. Hence an early stream of research into extensions of Gittins' index result concerned models in which new bandits arrived for processing over time. Whittle [35] gave an account of an index result for what he termed *arm-acquiring bandits* while Weiss [33] referred to *branching bandits*. Bertsimas and Niño-Mora [3] gave an account of index results for branching bandits from the perspective of the achievable region approach. Essentially, for such models whenever a bandit is processed it gives birth to a random collection of offspring bandits with probabilistic dynamics which are stationary but which are, however, allowed to depend on the state of the bandit processed. This certainly paves the way for consideration of models in which new bandits arrive over time in Poisson streams. Many such models have considered a time average cost criterion instead of one based on discounted rewards. The classical index result of Klimov [22] can be recovered in this manner. Nash [23] described how a range of such models could be obtained by taking the model $\{B_k, 1 \leq k \leq K\}$ and giving to each bandit B_k its own internal decision structure. When this is done bandits become

superprocesses. For a collection of such superprocesses, a policy must specify not only which bandit should be chosen for processing at each epoch but *how* it should be processed, namely which of its admissible internal actions should be taken. Whittle [34] gave an index result for such collections of superprocesses.

Whittle [36] progressed index theory considerably by proposing a *restless bandit* (RB) model which modified the Gittins MAB $\{B_k, 1 \leq k \leq K\}$ as follows: (i) the Markovian law P_k for evolution of the state of B_k under processing was replaced by the pair $\{P_k^1, P_k^2\}$, with P_k^1 determining state evolution under processing (the *active* action) and P_k^2 state evolution under the *passive* action; (ii) $M(< K)$ bandits are chosen for processing at each epoch; (iii) policies for bandit activation are evaluated under a time average reward criterion. Whittle's approach was to develop an index-based solution to the problem in which M bandits are chosen for processing *on average* (i.e. over time) via a Lagrangian relaxation. The indices so developed, commonly referred to as *Whittle indices*, extend those of Gittins in that they reduce to Gittins indices in the MAB case. However, Whittle indices only exist under conditions of *indexability*. This is a structural requirement of individual bandits and when applied to B_k means that when we consider the problem $B_k(\nu)$ in which an optimal processing policy is sought for B_k when a fixed charge ν is levied whenever processing takes place then it will be true that the set of states in which the active action (processing) is optimal decreases as ν increases. The Whittle index $\nu_k(x_k)$ then becomes a break-even value or fair charge and is the value of ν for which both active and passive actions are optimal when B_k is in state x_k . However natural indexability might appear to be, it is not universal and research continues to understand and characterise it more fully. Niño-Mora [25] has contributed extensively via the achievable region approach. In the literature direct approaches which investigate Whittle's Lagrangian relaxation to establish indexability and derive indices have resulted in the development of strongly performing index heuristics in many application areas. These include contributions to asset management (Glazebrook, Kirkbride and Ruiz-Hernandez [16]), inventory routing (Archibald, Black and Glazebrook [2]) and queueing control (Ansell, Glazebrook, Niño-Mora and O'Keeffe [1] and Glazebrook, Kirkbride and Ouenniche [15]). In the last quoted paper, it was possible to develop bounds on the closeness to optimality of the Whittle index heuristic and thereby to develop accounts of the policy's optimality in a range of asymptotic regimes. Whittle himself conjectured that, under the condition of indexability, his index heuristic was optimal in an asymptotic regime in which M and K diverge to infinity in fixed proportion. Weber and Weiss [32] gave a sufficient condition for this to be true and argued that even when the condition fails to hold the degree of suboptimality is very small.

Finally, reference has already been made to the achievable region approach to Gittins indexation pioneered *inter alia* by Tsoucas [29] and Bertsimas and Niño-Mora [3]. While the theory for MABs and for branching bandits can be developed around an exact characterisation of the associated achievable region, the approach facilitates progress with more complex models in which only a tight relaxation is available. Hence the approach has made possible the development of analytical performance bounds for index policies in a range of contexts together with proofs of asymptotic optimality. See, for example, the contributions of Dacre, Glazebrook and Niño-Mora [5], Glazebrook and Niño-Mora [18] and Dacre and Glazebrook [4] to the dynamic control of multi-class queueing systems and those of Glazebrook and Wilkinson [19] and Dunn and Glazebrook [6, 7] which concern MABs on parallel processors. For other contributions to

index theory which exploit the achievable region approach, see Garbe and Glazebrook [8, 9].

The reader should note that a full account of index theory by Gittins, Glazebrook and Weber [11] has recently been published and gives further details on the above contributions and many others. We now proceed in the next sections to discuss important new developments of index theory which extend existing work to embrace more complex resource configurations. The approach utilises the Lagrangian relaxation approach first espoused within index theory by Whittle [36].

3 A class of dynamic resource allocation problems

We present a model for the dynamic allocation of a single divisible resource to a collection of K projects denoted by $\{B_k, 1 \leq k \leq K\}$. The single resource is available to the projects at a constant rate $R \in \mathbb{Z}^+$ over an infinite horizon. Project B_k is characterised by its *state space* Ω_k , assumed countable, its *cost function* $c_k : \Omega_k \rightarrow \mathbb{R}^+$ and its Markov transition law q_k which models state evolution of the project in continuous time under application of the resource.

The controller of the system observes the states of all constituent projects continuously. The *system state* is the vector of the states of individual projects. Decision epochs occur at time 0 and at the times of all subsequent transitions of the system state. At each decision epoch, the controller inspects the current system state $\mathbf{x} = \{x_1, x_2, \dots, x_K\}$ and chooses an *admissible action* $\mathbf{a} = \{a_1, a_2, \dots, a_K\}$, namely a resource level to be applied to each project within the global resource constraint. The admissible action set A is given by

$$A := \left\{ \mathbf{a}; a_k \geq 0, 1 \leq k \leq K \text{ and } \sum_{k=1}^K a_k \leq R \right\}.$$

Should action $\mathbf{a}$ be taken at time $t \in \mathbb{R}^+$ when the system is in state $\mathbf{x}$ then the next transition of the system will occur at time $t + T(\mathbf{a}, \mathbf{x})$, where $T(\mathbf{a}, \mathbf{x})$ is an exponential random variable with rate $\sum_k \sum_{y_k} q_k(y_k | a_k, x_k)$. Hence for any given action $\mathbf{a}$ the projects evolve independently with Markov law $q_k(\cdot | a_k, x_k)$ determining transitions for project k when resource level a_k is applied in state x_k . Finally, the system incurs costs at rate $\sum_k c_k(x_k)$ when in state $\mathbf{x}$. The goal of analysis is the development of policies, namely, rules for choosing actions, which will minimise the average rate at which the system incurs costs over an infinite horizon or which will come close to doing so.

Direct application of the standard approaches of dynamic programming (DP) is computationally infeasible for most problems of realistic size and so primary interest centres on the development of *heuristic policies* for dynamic resource allocation which are close to cost minimising. Glazebrook *et al.* [14], (GHK) have developed a class of *index heuristics* for problems in which all constituent projects satisfy a structural property which they call *full indexability*. We first explain this approach before proposing a modification aimed at expanding the scope of these ideas.

We begin the discussion by expressing the optimized cost as an aggregation of contributions from individual projects as follows:

$$C^{opt} = \min_{\mathbf{u} \in \mathbf{U}} \sum_{k=1}^K C_k(\mathbf{u}), \quad (2)$$

In (2) C^{opt} is the minimum achievable cost rate, $\mathbf{U}$ is the class of admissible stationary policies and $C_k(\mathbf{u})$ is the time average cost rate earned by project k under policy $\mathbf{u}$. We create a *Lagrangian relaxation* (L) of the optimization in (2) by (i) extending the class of policies from $\mathbf{U}$ to $\mathbf{U}'$, where in the latter actions are chosen in a stationary way from the expanded action set

$$A' := \{\mathbf{a}; 0 \leq a_k \leq R, 1 \leq k \leq K\}, \quad (3)$$

and (ii) penalising violations of (time average versions of) the resource constraints. Expressed differently, instead of enforcing the global resource constraint, the relaxation L charges for resource usage. We express the relaxation L by

$$C(\nu) = \min_{\mathbf{u} \in \mathbf{U}'} \sum_{k=1}^K \{C_k(\mathbf{u}) + \nu R_k(\mathbf{u})\} - \nu R. \quad (4)$$

In (4), $R_k(\mathbf{u})$ is the time average rate at which resource is consumed by project k under policy $\mathbf{u}$ and the Lagrange multiplier ν has an interpretation as a charge levied per unit of resource and per unit of time. It is immediate from the above definitions that

$$C(\nu) \leq C^{opt}, \nu \geq 0.$$

Moreover, there is a projectwise decomposition of the optimization problem in (4) such that an optimal policy for L is obtained by running an optimal policy for all projects in parallel. We express this decomposition by writing

$$C(\nu) = \sum_{k=1}^K C_k(\nu) - \nu R,$$

where

$$C_k(\nu) := \min_{u_k \in U_k} \{C_k(u_k) + \nu R_k(u_k)\}, 1 \leq k \leq K, \quad (5)$$

is the minimised cost, inclusive of resource charges, obtainable from setting resource levels dynamically for project k only. In what follows, we shall denote the optimization problem in (5) by $B_k(\nu)$. In passing from the exact optimization problem in (2) to its relaxation in (4) we have replaced a single K -dimensional DP by K one-dimensional DPs. We conclude this section by noting that

$$\sup_{\nu \geq 0} C(\nu) \quad (6)$$

often constitutes a reasonably tight and easily computed lower bound for C^{opt} .

In order to develop simple index heuristics for the class of dynamic resource allocation problems under discussion we require the existence of optimal policies for the problems $B_k(\nu)$ which, for each project B_k , are monotone decreasing in ν (the higher the resource charge, the less resource used) in the following strong sense:

Definition 1 Project B_k is **fully indexable** if there exist stationary policies $\{u_k(v), \nu \in \mathbb{R}^+\}$ such that (i) $u_k(v)$ is optimal for $B_k(\nu)$, and (ii) $u_k(x, v)$, the resource used by policy $u_k(v)$ in state x , is decreasing in $\nu \forall x \in \Omega_k$.

The existence of optimal policies $u_k(v)$ for which the average rate of resource consumption $R_k\{u_k(v)\}$ is decreasing in ν is easy to establish. The requirement in the foregoing definition, namely that the monotonicity in ν should be uniform over project states, is much stronger and is not guaranteed. A fully indexable project k together with its qualifying policies $\{u_k(v), \nu \in \mathbb{R}^+\}$ yields a set of project indices as follows:

Definition 2 If project B_k is fully indexable with qualifying policies $\{u_k(v), \nu \in \mathbb{R}^+\}$, its corresponding project indices are given by

$$\nu_k(a, x) = \inf \left\{ \nu \in \mathbb{R}^+; u_k(x, v) \leq a \right\}, 0 \leq a \leq R-1, x \in \Omega_k.$$

Hence the index $\nu_k(a, x)$ is the smallest resource charge which forces the optimal action (resource level) for project k to be at or below a when in state x . It may be naturally thought of as a *fair charge* for increasing project k 's resource level above a when in x . Note that it follows from the above definitions that $\nu_k(a, x)$ must be decreasing in a over its range $\forall k, x$. Hence the fair charge for increasing the resource allocated to a fully indexable project decreases as the current resource level increases. These ideas thus carry with them a natural law of diminishing returns. The following result is an immediate consequence of the foregoing definition.

Theorem 1 If all K projects are fully indexable the Lagrangian relaxation L is solved by the policy $\mathbf{u}(\nu)$ which runs the project-specific optimal policies $u_k(v), 1 \leq k \leq K$, in parallel and which is such that $\forall \mathbf{x} \in \mathbf{X}_{k=1}^K \Omega_k$

$$\mathbf{u}(\mathbf{x}, \nu) = \mathbf{a} \Leftrightarrow \nu_k(a_k - 1, x_k) > \nu \geq \nu_k(a_k, x_k), 1 \leq k \leq K. \quad (7)$$

In words, the above theorem states that to develop optimal actions for L we should accumulate resource at each project until the fair charge for adding further resource falls below the actual (or *prevailing*) charge ν . This naturally leads on to a natural *greedy index* heuristic for fully indexable versions of the optimization problem (2) in which all indices are (strictly) positive. For such problems, in each system state $\mathbf{x}$, successive allocations of resource are made to projects in descending order of the above fair charges until all of the resource is exhausted. If we write $\mathbf{u}_\nu$ for the resulting policy, we have $\forall \mathbf{x} \in \mathbf{X}_{k=1}^K \Omega_k$

$$\mathbf{u}_\nu(\mathbf{x}) = \mathbf{a} \Rightarrow \nu_k(a_k - 1, x_k) \geq \nu_l(a_l, x_l), l \neq k, 1 \leq k \leq K, \text{ and } \sum_{k=1}^K a_k = R. \quad (8)$$

Please note that, should the action specified in (7) consume R units of resource then it will coincide with that in (8). In fully indexable problems in which some of the fair charges are zero, the greedy index heuristic $\mathbf{u}_\nu$ defined above is modified by the requirement that no allocations of resource corresponding to zero fair charges are made. When this is the case, it may be that $\mathbf{u}_\nu$ allocates less than R units of resource in some system states.

Example 1 Consider a model in which $\Omega_k = \mathbb{N}, c_k(x) = c_k x$ and $q_k(y | a, x) = \lambda_k I(y = x + 1) + \mu_k(a) I(y = x - 1, x \neq 0), 1 \leq k \leq K$, for all choices of a, x and y . Here and elsewhere $I(\cdot)$ is an indicator. Such a model was considered by GHK and represents a situation in which a team of size R is available to provide service at K stations at which new customers arrive in independent Poisson streams, with λ_k the

rate at station k . The continuously observed system state is the vector of station head counts and a linear holding cost is incurred at each. At each point in time the team is divided between the stations in response to demand. When a servers are allocated to station k a service rate $\mu_k(a)$ is delivered, where $\mu_k : \{0, 1, \dots, R\} \rightarrow \mathbb{R}^+$ is increasing and concave. If conditions are imposed which guarantee the existence of stable policies then all stations can be shown to be fully indexable. In an extensive numerical study the greedy index heuristic $\mathbf{u}_\nu$ performs outstandingly well.

Example 2 Consider a model in which $\Omega_k = \{-M_k, -M_k + 1, \dots, N_k - 1, N_k\}$ where $M_k, N_k \in \mathbb{Z}^+$, $c_k(x) = h_k x^+ + b_k x^- + l_k \mu_k I(x = -M_k)$ and $q_k(y | a, x) = \lambda_k(a) I(y = x + 1, x \neq N_k) + \mu_k I(y = x - 1, x \neq -M_k)$, $1 \leq k \leq K$, for all choices of a, x and y . Such a model was considered by Hodge and Glazebrook [20] and represents a situation in which some resource (manpower, machinery) is available to drive the manufacture of K products to stock. The resulting inventories are available to meet exogenous demand, which for product k occurs at Poisson rate μ_k . The maximum amount of product k which may be held in stock is N_k while the maximum number of product k backorders at any time is M_k . Once this backorder limit is achieved further demand for the product results in lost sales. The cost of holding a single unit of product k in stock for one unit of time is h_k , while b_k is the cost of a single product k backorder per unit of time and l_k is the cost of a single lost sale for k . When a units of resource are applied to the manufacture of product k , a production rate $\lambda_k(a)$ results, where $\lambda_k : \{0, 1, \dots, R\} \rightarrow \mathbb{R}^+$ is increasing and concave. Hodge and Glazebrook [20] show that there exists an easily computed positive number h_k^* such that $h_k \leq h_k^*$ guarantees product k 's full indexability. Since this sufficient condition covers most holding cost rates that are likely to be encountered in practice, we have a form of practical indexability. When all products are fully indexable an extensive numerical study provides evidence that the greedy index heuristic $\mathbf{u}_\nu$ performs outstandingly well. The authors go on to discuss the consequences of any failure of full indexability to hold and demonstrate how carefully designed heuristics based on fair charges can still perform well.

These two examples notwithstanding, model classes in which it has proved possible to establish full indexability have been limited to those in which project evolution is of birth-death type, namely project state transitions are to neighbouring states only. The DP analyses on which the proofs have rested are intricate and lean heavily on this simplicity of structure. In light of this challenge, in the next section we develop an approximative approach to the above index ideas which have broader applicability.

4 Approximate indices for dynamic resource allocation

Given that the requirements of Definition 1 above are too exacting for practical implementation in many contexts we resume the discussion by relaxing the requirements of full indexability.

Definition 3 (i) The family of policies $F_k \equiv \{u_k^F(v), \nu \geq 0\}$ is *index generating for project* B_k if (a) $u_k^F(v)$ is stationary for $B_k(\nu)$, and (b) $u_k^F(x, v)$, the resource used by policy $u_k^F(v)$ in state x , is decreasing in $\nu \forall x \in \Omega_k$. (ii) If the family of policies F_k is index generating for project B_k , the index generated by F_k is ν_k^F :

$\{0, 1, \dots, R-1\} \times \Omega_k \rightarrow \mathbb{R}^+$ and is given by

$$\nu_k^F(a, x) = \inf \left\{ \nu \in \mathbb{R}^+; u_k^F(x, \nu) \leq a \right\}, 0 \leq a \leq R-1, x \in \Omega_k.$$

If $u_k^F(x, \nu) > a$, whenever $\nu \geq 0$, then $\nu_k^F(a, x) = \infty$.

Hence we recover the ideas of full indexability of the previous section when the index generating family F_k consists of optimal policies for the family of problems $\{B_k(\nu), \nu \in \mathbb{R}^+\}$. What we shall instead do in this section is look for strongly performing (though not necessarily optimal) families of policies for these problems which are index generating. If we can do this for all constituent projects, then running such policies in parallel creates strongly performing policies for the above Lagrangian relaxation L which are of index form as in Theorem 1 above. However, the indices concerned will now be those generated by the index generating families $F_k, 1 \leq k \leq K$. In this way we develop greedy index heuristics $\mathbf{u}_\nu^F$ for the dynamic resource allocation problem in (2), structured such that

$$\mathbf{u}^F(\mathbf{x}, \nu) = \mathbf{a} \Rightarrow \nu_k^F(a_k - 1, x_k) \geq \nu_l^F(a_l, x_l), l \neq k, 1 \leq k \leq K, \text{ and } \sum_{k=1}^K a_k = R,$$

with the above proviso that resource is never allocated in response to zero index values. One method for constructing strongly performing index generating policies which we have found to be effective proceeds in two stages: (i) at stage 1 construct an optimal policy for $B_k(\nu)$ within the class of *static* policies which choose a fixed level of resource for all project states. When we perform this parametric optimisation the range of possible resource levels is usually expanded to the closed interval $[0, R]$ or perhaps $\mathbb{R}^+$ and hence may be inadmissible since integer resource levels no greater than R are no longer required. This inadmissibility is not a problem since it is corrected at stage 2; (ii) at stage 2 construct an admissible stationary and dynamic policy for $B_k(\nu)$ by imposing a single DP policy improvement step on the policy determined at stage 1. We pause to note that this approach (of imposing a single policy improvement step on an optimal static policy) has been found to be an effective route to the development of strongly performing policies in a number of application contexts. We now illustrate our approach with reference to two examples, neither of which are of birth-death type and hence which have proved beyond any demonstration of full indexability to date.

5 The optimal allocation of a team of servers to parallel queues with batch arrivals

We develop the fully indexable model discussed as Example 1 above by allowing batches of arrivals at the service stations. Hence we now have arrivals in the form of independent compound Poisson streams with λ_k the rate of batch arrivals at k and $\{p_n^k, n \geq 1\}$ the probability mass function for the size of station k arrival batches. In our general model class we make the choices $\Omega_k = \mathbb{N}, c_k(x) = c_k x$ and $q_k(y | a, x) = \lambda_k p_n^k I(y = x + n) + \mu_k(a) I(y = x - 1, x \neq 0), 1 \leq k \leq K$, for all choices of a, n, x and y . Other features of the problem are as above including the assumption that $\mu_k : \{0, 1, \dots, R\} \rightarrow \mathbb{R}^+$ is increasing and concave. In summary, our goal is the allocation of a team of R servers to a collection of stations receiving batches of arrivals to minimise the rate of linear

costs incurred by customers waiting at the stations. The service rate at each station is increasing and concave in the number of servers allocated there and so there is a law of diminishing returns associated with increasing the level of resource available locally. This should mean that policies which perform well will tend to spread the service resource around.

Since ideas of indexability and indices relate to individual stations we focus on a single station and drop the identifier k . Hence the station will be characterised by its cost rate c , batch arrival rate λ , batch size distribution $\{p_n, n \geq 1\}$ and increasing and concave service rate function $\mu(\cdot)$. It will be convenient to write the mean batch size and its variance by

$$\sum np_n = \theta, \sum (n - \theta)^2 p_n = \sigma^2.$$

Plainly $\theta > 1$. We shall refer to station problem $B(\nu)$.

We now attempt to implement our two stage approach to develop an index generating family of policies for the service station. Recall that at stage 1 we attempt to develop an optimal policy from a static class which operates a fixed resource level in all states. By a modest abuse of notation we write a for the station policy such that $a(x) = a, x \in \Omega$, namely which fixes the resource available to the station at level a . Firstly, note that standard results in queueing theory indicate that we can restrict a to the range for which $\mu(a) > \lambda\theta$ since otherwise the queue at the station will fail to be stable and the average cost rate for $B(\nu)$ will be infinite. Hence we need the additional assumption that $\hat{A} \equiv \{a; a \in \mathbb{R}^+ \text{ and } \mu(a) > \lambda\theta\} \neq \emptyset$. Under the static policy $a \in \hat{A}$, it is not difficult to show that the time average holding cost at the station is given by

$$\gamma(a) \equiv \frac{c\lambda\mu(a)(\theta + \theta^2 + \sigma^2)}{2\{\mu(a) - \lambda\theta\}\{\lambda + \mu(a) - \lambda\theta\}}, a \in \hat{A}, \quad (9)$$

and hence an optimal static policy for problem $B(\nu)$ is a solution to the problem

$$\inf_{a \in \hat{A}} (\gamma(a) + \nu a). \quad (10)$$

We shall refer to any minimiser of (10) by $a(\nu)$.

Lemma 1 *If, in addition to the above conditions, $\mu(\cdot)$ is twice continuously differentiable, strictly increasing and strictly concave then $a(\nu)$ is unique and is decreasing in ν .*

Proof The result is a straightforward consequence of the fact that the conditions in the lemma on service rate function $\mu(\cdot)$ imply that the quantity $\gamma(a)$ is twice differentiable, strictly decreasing and strictly convex in a .

We now proceed to the second stage of our attempt to construct an index generating family of policies for the station. This involves strengthening the static policy $a(\nu)$ by implementing a single DP policy improvement step. Equivalently, having observed the station state (head count) x we choose an action (resource level) at the station to minimise holding costs under an assumption that *all future resource levels* will be determined by the static policy $a(\nu)$. We make this idea more precise in what follows.

We first need to develop some measures relating to the evolution of the station state under general static policy $a \in \hat{A}$. Suppose that $X(0) = x$, namely that the station evolves under policy a from state x at time 0. We wish to develop a strong

approximation to $C(a, x, t)$, the expected holding cost incurred at the station over the time period $[0, t]$. By standard theory, we have that $C(a, x, t) \sim t\gamma(a) + b(a, x)$ where $b(a, x)$ is the *bias*, namely the transient effect associated with starting the system in state x . In order to compute the bias, we need the quantity $C(a, x)$, the expected holding cost incurred until the station, evolving from initial state x under policy a , empties for the first time, and the quantity $T(a, x)$, the corresponding expected time. By standard arguments from queueing theory we have that

$$C(a, x) = \frac{cx(x-1)}{2\{\mu(a) - \lambda\theta\}} + \frac{cx(2\mu + \lambda\theta^2 + \lambda\sigma^2 - \lambda\theta)}{2\{\mu(a) - \lambda\theta\}^2} \quad (11)$$

and

$$T(a, x) = \frac{x}{\mu(a) - \lambda\theta},$$

with the bias being given by

$$b(a, x) = C(a, x) - \gamma(a)T(a, x). \quad (12)$$

In order to implement the one-step DP policy improvement step, suppose that $X(0) = x \geq 1$, that action $a \in \{0, 1, \dots, R\}$ is implemented at station k from 0 up to time $\min\{T, S\} \sim \exp\{\lambda + \mu(R)\}$ where $T \sim \exp\{\lambda + \mu(a)\}$ is the time of the first transition of the system and $S \sim \exp\{\mu(R) - \mu(a)\}$ is independent of S . Beyond $\min\{T, S\}$ the static policy $a(\nu)$ is implemented for a further t time units. Both holding costs and resource charges are incurred. The problem of choosing action a to minimise the total expected cost incurred during $[0, \min\{T, S\} + t]$ can be expressed as

$$\min_{a \in \{0, 1, \dots, R\}} (\nu a - \mu(a) [C\{a(\nu), x, t\} - C\{a(\nu), x-1, t\}]). \quad (13)$$

If we now let $t \rightarrow \infty$ in (13) then the dynamic policy u^F designed by our two stage procedure is given by

$$\begin{aligned} u^F(x, \nu) &= I(x \geq 1) \operatorname{argmin}_{a \in \{0, 1, \dots, R\}} (\nu a - \mu(a) [b\{a(\nu), x\} - b\{a(\nu), x-1\}]) \\ &= I(x \geq 1) \operatorname{argmin}_{a \in \{0, 1, \dots, R\}} \left(\nu a - \right. \\ &\quad \left. \mu(a) \left[\frac{cx}{\mu\{a(\nu)\} - \lambda\theta} + \frac{c\lambda^2(1-\theta)(\theta^2 + \sigma^2 + \theta)}{2(\mu\{a(\nu)\} - \lambda\theta)^2(\mu\{a(\nu)\} - \lambda\theta + \lambda)} \right] \right), \quad (14) \end{aligned}$$

where (14) uses (9), (11), (12) and (13).

By the strict concavity of the service rate function $\mu(\cdot)$, any optimising a in (14) will be as described in the following result which is given without proof and follows straightforwardly from the preceding calculations. In the statement of the lemma we use the quantity

$$K(x, \nu) = cx + \frac{c\lambda^2(1-\theta)(\theta^2 + \sigma^2 + \theta)}{2(\mu\{a(\nu)\} - \lambda\theta)(\mu\{a(\nu)\} - \lambda\theta + \lambda)}, \nu \geq 0, x \geq 1.$$

Lemma 2 *The policy u^F for $B(\nu)$ obtained by one-step policy improvement from the optimal static policy $a(\nu)$ satisfies*

$$u^F(x, \nu) = a \Rightarrow \{\mu(a) - \mu(a-1)\} \frac{K(x, \nu)}{\nu(\mu\{a(\nu)\} - \lambda\theta)} \geq 1 \geq \{\mu(a+1) - \mu(a)\} \frac{K(x, \nu)}{\nu(\mu\{a(\nu)\} - \lambda\theta)}, x \geq 1,$$

where we take $\mu(-1) = -\infty$ and $\mu(R+1) = \mu(R)$.

We are now ready to state the main result of this section.

Theorem 2 *If the conditions of Lemma 1 are satisfied then the family of policies $\{u^F(\nu), \nu \geq 0\}$ is index generating and ν^F , the index generated, is such that $\nu^F(a, x)$ is, for each $x \geq 1$, the unique positive solution in ν of the equation*

$$\frac{\{\mu(a+1) - \mu(a)\} K(x, \nu)}{\nu(\mu\{a(\nu)\} - \lambda\theta)} = 1. \quad (15)$$

should one exist. If there is no positive solution to (15) then $\nu^F(a, x) = 0$.

Proof Lemma 1 asserts that $a(\nu)$ is decreasing in ν from which it easily follows that $K(x, \nu)$ is strictly decreasing in ν for each fixed $x \geq 1$. Plainly from Lemma 2 if $K(x, \nu) < 0$ then $u^F(x, \nu) = 0$. It is now straightforward from Lemma 2 that, in order for the family $\{u^F(\nu), \nu \geq 0\}$ to be index generating it is enough that $\nu(\mu\{a(\nu)\} - \lambda\theta)$ is increasing in ν . However, it is trivial from (9) and (10), following some trivial algebraic manipulation, that

$$\nu(\mu\{a(\nu)\} - \lambda\theta) = \frac{c\lambda(\theta^2 + \sigma^2 + \theta)}{2} \mu'\{a(\nu)\} \times \left[\frac{\lambda\theta}{(\mu\{a(\nu)\} - \lambda\theta)(\mu\{a(\nu)\} - \lambda\theta + \lambda)} + \frac{1}{(\mu\{a(\nu)\} - \lambda\theta + \lambda)} + \frac{\lambda(\theta - 1)}{(\mu\{a(\nu)\} - \lambda\theta + \lambda)^2} \right].$$

It is now trivial from Lemma 1 together with the strictly increasing and strictly concave nature of $\mu(\cdot)$ that $\mu\{a(\nu)\}$ is decreasing and $\mu'\{a(\nu)\}$ is increasing in ν and hence that $\nu(\mu\{a(\nu)\} - \lambda\theta)$ is increasing in ν . The claimed form of the index ν^F is trivial. This completes the proof.

Remark 1 It is easy to show that $\nu^F(a, x)$ is increasing in x for each fixed a as is natural in this case. Hence under any greedy index policy using these indices, more resource will be directed toward longer queues.

As an illustration of the performance of the natural greedy index heuristic arising from our newly created index generating family, find in Table 1 a summary of statistics obtained by computer simulation for the batch arrival model of this section. For our study a suitable collection of problems was chosen with the following characteristics:

- a batch arrival distribution, common to all stations, with $p_1 = 0.1$, $p_2 = p_3 = p_4 = 0.3$, with mean 2.8;
- a range of numbers of stations, K , from 2 to 5;
- a range of values of total available resource, from $3K - 2$ to $3K + 2$.

For each problem instance the arrival rates at the stations, λ_k , were drawn independently from a uniform distribution on $(0.8, 1.1)$. The concave service rate function used at station k was $\mu_k(a) = \mu_k(5 - 0.2(5 - a)^2)$ for $a = 0, 1, 2, \dots, 5$, where the μ_k were drawn uniformly at random from $(0, 9, 1.1)$. Each station is maximally allowed five units of resource at any time.

We refer to the greedy index heuristic policy proposed in this section as OSPI (one-step policy improvement). In addition to analysing OSPI, we also present the performance of the greedy index heuristic derived from the best static solution $a(\nu)$, namely STATIC. Furthermore we present the performances of three additional natural heuristic policies. One, which we call FIX, is the best fully static policy, with no dynamic change of policy as the system state changes. Another, which we call MYOPIC, chooses actions dynamically to maximize the rate at which service is delivered. Finally, we consider a version of the $c\mu$ -rule, known to be optimal in the birth-death single resource problem, and which we call CMU*. To extend this rule to divisible resources we assume an ‘all or nothing’ approach to the resource level at any given station. For our problems, with a five unit limit at each station, this is equivalent to greedily and dynamically allocating as much resource as is available to non-empty stations in decreasing order of the value of $c_k(x_k)\mu_k(5)$, up to our maximum of five units per station. We also use suitable versions of this variety of alternative heuristics in Section 6 when we study a machine maintenance model.

	STATIC	OSPI	FIX	MYOPIC	CMU*
MIN	-3.3637	—	3.7743	-3.5698	0.2208
LQ	0.5685	—	23.6858	1.0403	9.4270
MED	1.0912	—	4648.7113	1.9942	29.7569
UQ	1.7012	—	$> 10^4$	3.1877	66.6442
MAX	56.7365	—	$> 10^4$	60.8797	27684.5893
Mean Cost	9.8118	9.6962	762.2770	9.9609	118.6721

Table 1 Batch Arrivals: 32,000 scenarios. Percentage cost excess over OSPI. See details in text.

For each number of stations K , $2 \leq K \leq 5$, 2,000 problems were randomly generated with characteristics described above, and through Monte-Carlo simulation very good estimates of the associated cost rates were obtained. In Table 1 are presented the percentage extra costs resulting from using each policy described above, with our improved index policy (OSPI) used as the benchmark. We see that occasionally OSPI is beaten by STATIC or MYOPIC but over the vast majority of problems it yields at least a modest reduction in costs. The more substantial gains over STATIC and MYOPIC were obtained for two-station problems when overall load was close to available capacity. In those cases the inability to dynamically adjust resource levels results in enormous suboptimality in FIX. The same scenarios are responsible for large ($\approx 50\%$) suboptimality of STATIC and MYOPIC where even these dynamic policies are heavily outperformed by OSPI. A more extensive study has been conducted, including using other service rate functions, the details of which will follow in an extended paper. From this we can also see that moderate sized gains from OSPI tend to result for two-station problems where one station evolves faster than the other. In total 32,000 sample scenarios are summarised in Table 1. For each choice of K , four total resource

level choices: $3K - 1$, $3K$, $3K + 1$ and $3K + 2$ were considered, with 2,000 problems chosen for each.

6 On the optimal maintenance of a collection of machines

We now consider a situation in which K stochastic projects deteriorate independently in the absence of any application of resource. This deterioration is modelled as drift toward higher value states in which costs are incurred at higher rates. The application of resource to a project can arrest decline by returning it to the pristine state 0. We thus generalise the restless bandit models of machine maintenance discussed by Glazebrook *et al.* [17]. We make the following modelling choices: the project k state space Ω_k is $\mathbb{N}$ and its cost rate is given by $c_k(x) = c_k x + d_k x(x-1)$, $x \geq 0$. Project k transition rates are given by $q_k(y | a, x) = \lambda_k(a) I(y=0) + \mu_k I(y=x+1)$ for all choices of a, x and y where μ_k is project k 's deterioration rate while $\lambda_k(a)$ is, in the machine maintenance context, the repair rate experienced by k when resource level a is applied. We shall assume that each $\lambda_k(\cdot)$ is increasing and concave.

To discuss issues of indexability, we consider a single project/machine and drop the identifier k from the notation. Our focus is on the problem $B(\nu)$ which concerns the optimal setting of resource levels for the project when resource has to be paid for at rate ν per unit of time and per unit of resource. As before, we first consider the project operating under constant resource level $a \in \mathbb{R}^+$. It is straightforward to show that the time average project costs under static policy a is given by

$$\gamma(a) = \frac{c\mu}{\{\lambda(a)\}^2} + \frac{2d\mu^2}{\{\lambda(a)\}^3}.$$

We denote by $a(\nu)$ the solution of the static problem for (ν) , namely

$$a(\nu) = \operatorname{argmin}_{a \in \mathbb{R}^+} (\gamma(a) + \nu a)$$

The following is straightforward to establish.

Lemma 3 *If, in addition to the above conditions, $\lambda(\cdot)$ is twice continuously differentiable, strictly increasing and strictly concave then $a(\nu)$ is unique and is decreasing in ν .*

We denote by $C(a, x)$ the expected total cost incurred (excluding resource costs) in returning the project to pristine state 0 from initial state x under the continuous application of action $a \in \mathbb{R}^+$. We also use $T(a, x) = (\lambda(a))^{-1}$ for the corresponding expected time. We have

$$C(a, x) = \frac{cx}{\lambda(a)} + \frac{c\mu}{\{\lambda(a)\}^2} + \frac{dx(x-1)}{\lambda(a)} + \frac{2dx\mu}{\{\lambda(a)\}^2} + \frac{2d\mu^2}{\{\lambda(a)\}^3}.$$

The corresponding bias is given by

$$\begin{aligned} b(a, x) &:= C(a, x) - \gamma(a) T(a, x) \\ &= \frac{cx}{\lambda(a)} + \frac{dx(x-1)}{\lambda(a)} + \frac{2dx\mu}{\{\lambda(a)\}^2}. \end{aligned}$$

We now implement a one-step policy improvement in enhancement of static policy $a(\nu)$ to obtain the dynamic resource allocation policy $u^F(\nu)$ for the problem $B(\nu)$. Using similar calculations to those in the previous example, we deduce that

$$\begin{aligned} u^F(x, \nu) &= \operatorname{argmin}_{a \in \{0, 1, \dots, R\}} [\nu a - \lambda(a) b\{a(\nu), x\}] \\ &= \operatorname{argmin}_{a \in \{0, 1, \dots, R\}} \left(\nu a - \lambda(a) \left(\frac{cx}{\lambda\{a(\nu)\}} + \frac{dx(x-1)}{\lambda\{a(\nu)\}} + \frac{2dx\mu}{[\lambda\{a(\nu)\}]^2} \right) \right). \end{aligned}$$

We omit the proof of the following result:

Theorem 3 *If the conditions of Lemma 3 are satisfied then the family of policies $\{u^F(\nu), \nu \geq 0\}$ is index generating and ν^F , the index generated, is such that $\nu^F(a, x)$ is, for each $x \geq 0$, the unique positive solution in ν of the equation*

$$\{\lambda(a+1) - \lambda(a)\} b\{a(\nu), x\} = \nu.$$

Remark 2 It is easy to show that $\nu^F(a, x)$ is increasing in x for each fixed a as is natural in this case. Hence under any greedy index policy using these indices, more resource will be directed toward machines in the poorest condition.

As explained in the numerical study accompanying the batch arrival model in Section 5, we have implemented this greedy index policy, created via one-step policy improvement (OSPI), for the machine maintenance scenario. We have also assessed computationally the performances of rival heuristics: STATIC, FIX, MYOPIC and CMU*, the same competitor policies considered in Section 5. All but FIX are dynamic resource allocation heuristics, though as we again see, now in Table 2, our proposal OSPI clearly outperforms the rival policies.

In Table 2 we present the summary statistics (percentage cost excess over OSPI) from a total of 6000 problems. The underlying model parameters, such as the number of stations, are kept from the previous numerical study of batch arrivals. Thus for each choice K , the total number of stations, $2 \leq K \leq 5$, a variety of total resource levels were selected, in particular $3K - 2$, $3K$ and $3K + 2$, and 500 problems generated from each.

	STATIC	OSPI	FIX	MYOPIC	CMU*
MIN	2.1200	—	7.3815	2.2998	-0.6366
LQ	3.6707	—	21.5691	4.2289	1.5976
MED	6.2518	—	31.2932	6.5127	4.9011
UQ	8.8725	—	$> 10^7$	9.3070	6.4934
MAX	14.1095	—	$> 10^7$	16.8721	28.4565
Mean Cost	62.3510	58.4785	$> 10^7$	62.7378	62.2484

Table 2 Machine maintenance: 6,000 scenarios. Percentage cost excess over OSPI. See details in text.

For these problems, as in the batch arrival problems previously considered, it is clear that none of the rival heuristics are on par with OSPI. Outperforming STATIC, MYOPIC and CMU* by 5% is the norm for OSPI, with occasionally much larger gains. Again a wide range of problems were considered. For the purposes of our calculations

the alternative service rate function $\mu_k(a) = \mu_k a / (a + \alpha)$ was used with the α_k drawn uniformly at random from (1.8, 2.0) for each station in every scenario, along with cost function parameters $c_k = 1$, $d_k = 4$. A larger numerical study has also been conducted for these problems. Further details of this will appear in an extended paper. In summary, the story is the same regarding relative performance of the policies.

Acknowledgements The first two authors acknowledge the support for this work by the Engineering and Physical Sciences Research Council (EPSRC) through grant EP/E049265/1. The third author was supported by an RCUK fellowship and the fourth author by an EPSRC doctoral studentship.

References

1. P.S. Ansell, K.D. Glazebrook, J. Niño Mora, and M. O’Keeffe. Whittle’s index policy for a multi-class queueing system with convex holding costs. *Math. Meth. Oper. Res.*, 57:21–39, 2003.
2. T. W. Archibald, D. P. Black, and K. D. Glazebrook. Indexability and index heuristics for a simple class of inventory routing problems. *Oper. Res.*, 57:314–326, 2009.
3. D. P. Bertsimas and J. Niño Mora. Conservation laws, extended polymatroids and multi-armed bandit problems: a polyhedral approach to indexable systems. *Math. Oper. Res.*, 21:257–306, 1996.
4. M. J. Dacre and K. D. Glazebrook. The dependence of optimal returns from multi-class queueing systems on their customer base. *QUESTA*, 40:93–115, 2002.
5. M. J. Dacre, K. D. Glazebrook, and J. Niño Mora. The achievable region approach to the optimal control of stochastic systems (with discussion). *J. R. Statist. Soc.*, B61:747–791, 1999.
6. R. T. Dunn and K. D. Glazebrook. The performance of index-based policies for bandit problems with stochastic machine availability. *Adv. App. Prob.*, 33:365–390, 2001.
7. R. T. Dunn and K. D. Glazebrook. Discounted multi-armed bandit problems on a collection of machines with varying speeds. *Math. Oper. Res.*, 29:266–279, 2004.
8. R. Garbe and K. D. Glazebrook. Stochastic scheduling with priority classes. *Math. Oper. Res.*, 23:119–144, 1998.
9. R. Garbe and K. D. Glazebrook. Submodular returns and greedy heuristics for queueing scheduling problems. *Oper. Res.*, 46:336–346, 1998.
10. J. C. Gittins. Bandit processes and dynamic allocation indices (with discussion). *J. R. Statist. Soc.*, B41:148–177, 1979.
11. J. C. Gittins, K. D. Glazebrook, and R. R. Weber. *Multi-armed Bandit Allocation Indices*. Wiley-Blackwell, London, 2nd edition, 2011.
12. J. C. Gittins and D. M. Jones. A dynamic allocation index for the sequential design of experiments. In *Progress in Statistics*, pages 241–266. North-Holland, Amsterdam, 1974.
13. K. D. Glazebrook. Stochastic scheduling with order constraints. *Int. J. Syst. Sci.*, 7:657–666, 1976.
14. K. D. Glazebrook, D. J. Hodge, and C. Kirkbride. General notions of indexability for queueing control and asset management. *Ann. App. Prob.*, 2011. (to appear).
15. K. D. Glazebrook, C. Kirkbride, and J. Ouenniche. Index policies for the admission control and routing of impatient customers to heterogeneous service stations. *Oper. Res.*, 57:975–989, 2009.
16. K. D. Glazebrook, C. Kirkbride, and D. Ruiz-Hernandez. Spinning plates and squad systems – policies for bi-directional restless bandits. *Adv. Appl. Prob.*, 38:95–115, 2006.
17. K. D. Glazebrook, H. M. Mitchell, and P. S. Ansell. Index policies for the maintenance of a collection of machines by a set of repairmen. *Eur. J. Oper. Res.*, 165:267–284, 2005.
18. K. D. Glazebrook and J. Niño Mora. Parallel scheduling of multiclass $M/M/m$ queues: approximate and heavy-traffic optimization of achievable performance. *Oper. Res.*, 49:609–623, 2001.
19. K. D. Glazebrook and D. J. Wilkinson. Index-based policies for discounted multi-armed bandits on parallel machines. *Ann. App. Prob.*, 10:877–896, 2000.
20. D. J. Hodge and K. D. Glazebrook. Dynamic resource allocation in a multi-product make-to-stock production system. *QUESTA*, 2011. (to appear).

21. M. N. Katehakis and A. F. Veinott. The multi-armed bandit problem – decomposition and computation. *Math. Oper. Res.*, 12:262–268, 1987.
22. G. P. Klimov. Time sharing systems I. *Theory Prob. Applic.*, 19:532–551, 1974.
23. P. Nash. *Optimal Allocation of Resources Between Research Projects*. PhD thesis, Cambridge University, 1973.
24. J. Niño Mora. Dynamic priority allocation via restless bandit marginal productivity indices. *TOP*, 15:161–198, 2007.
25. J. Niño-Mora. Restless bandits, partial conservation laws and indexability. *Adv. Appl. Prob.*, 33:76–98, 2001.
26. D. R. Robinson. Algorithms for evaluating the dynamic allocation index. *Oper. Res. Lett.*, 1:72–74, 1982.
27. I. M. Sonin. A generalized Gittins index for a Markov chain and its recursive calculation. *Statist. and Prob. Lett.*, 78:1526–1533, 2008.
28. J. N. Tsitsiklis. A short proof of the Gittins index theorem. *Ann. Appl. Prob.*, 4:194–199, 1994.
29. P. Tsoucas. The region of achievable performance in a model of Klimov. Technical report, IBM, 1991.
30. P. Varaiya, J. Walrand, and C. Buyukkoc. Extensions of the multi-armed bandit problem. *IEEE Trans. Autom. Control*, AC-30:426–439, 1985.
31. R. R. Weber. On the Gittins index for multiarmed bandits. *Ann. Appl. Prob.*, 2:1024–1033, 1992.
32. R. R. Weber and G. Weiss. On an index policy for restless bandits. *J. Appl. Prob.*, 27:637–648, 1990. (Addendum: *Adv. Appl. Prob.*, 23:429–430, 1991).
33. G. Weiss. Branching bandit processes. *Prob. Eng. Inform. Sci.*, 2:269–278, 1988.
34. P. Whittle. Multi-armed bandits and the Gittins index. *J. R. Statist. Soc.*, B42:142–149, 1980.
35. P. Whittle. Arm-acquiring bandits. *Ann. Prob.*, 9:284–292, 1981.
36. P. Whittle. Restless bandits: activity allocation in a changing world. In *A Celebration of Applied Probability*, pages 287–298. (*J. Appl. Prob. Spec. Vol. 25A*), ed. J. Gani, Applied Probability Trust, Sheffield, 1988.

Quasi-Elementary Landscapes

Darrell Whitley

Abstract There exists local search landscapes where the evaluation function is an eigenfunction of the Laplacian that describes the graph that corresponds to the neighborhood structure of the search space. Problems that display this structure are called “Elementary Landscapes” and they have a number of special mathematical properties. In this paper we show that there exists landscapes that are “almost” elementary. The landscapes can be described as being elementary with some corrections to describe those part of the neighborhood structure that deviate from the normal structure found in elementary landscapes. The term “Quasi-elementary landscapes” is introduced to describe these landscapes. Quasi-elementary can also be elementary for specific problem sizes, and not elementary for other problem sizes.

1 Introduction

Grover [4] originally observed that there exists neighborhoods for Traveling Salesman Problem (TSP), Graph Coloring, Min-Cut Graph Partitioning, Weight Partitioning, as well as Not-All-Equal-SAT that can be modeled using a wave equation borrowed from mathematical physics. Stadler [7] named this class of problems “elementary landscapes” and showed that if a landscape is elementary, the objective function is an eigenfunction of the Laplacian matrix that describes the connectivity of the neighborhood graph representing the search space. If a landscape is elementary, it is possible to compute $\bar{f}$ and the average evaluation over all of the neighbors of any solution x in closed form; we denote the set of solutions that are neighbors of solution x by $N(x)$ and the neighborhood average is given by $\text{Avg}(N(x)) = \frac{1}{|N(x)|} \sum_{y \in N(x)} f(y)$. However, for an elementary landscape this average can also be computed as follows:

$$\text{Avg}(N(x)) = f(x) + \frac{k}{d}(\bar{f} - f(x))$$

where $-k$ is the eigenvalue of the Laplacian and d is the neighborhood size.

Department of Computer Science
Colorado State University
E-mail: whitley@cs.colostate.edu

Whitley and Sutton developed a component based model of elementary landscapes that makes it easy to identify elementary landscapes [13]. In most cases, the components are weights in a cost matrix. In the case of pseudo-Boolean functions, the components can also be the weight coefficients of a polynomial form of the cost function. Let the set of components be denoted by C ; a solution x will also denote the subset of components that contribute to the evaluation of x so that the sum of the components that contribute to x is the same as the evaluation of x , denoted by $f(x)$. Finally, let $C - x$ denote the subset of components that do not contribute to the evaluation of solution x . For the Traveling Salesman Problem, the solution x can also denote the subset of cost (distance) weights in solution x and $C - x$ will denote the subset of weights in the cost matrix that do not contribute to $f(x)$.

In the component model $k = \alpha + \beta$, where $p_1 = \alpha/d$ is the proportion of the components (weights) in solution x that change when moving from solution x to a neighboring solutions y , $d = |N(x)|$, and $p_2 = \beta/d$ is the rate at which components in the set $C - x$ are sampled when moving from solution x to a neighboring solutions y . Another precondition for an landscape to be elementary is:

$$\bar{f} = p_3 \sum_{c \in C} c$$

where p_3 is the sampling rate of components across all solutions as they contribute to $\bar{f}$.

The current paper looks at landscapes and neighborhoods that are not elementary, but which are “almost” elementary. The term *quasi-elementary landscape* is introduced to describe a landscape and neighborhood structure where the wave equation can be used to compute the neighborhood average by adding a small correction. Thus, a landscape is quasi-elementary if:

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) - p_1 f(x) + p_2 \left(\frac{\bar{f}}{p_3} - f(x) \right) + g(x) \quad (1)$$

We will refer to g as an auxiliary function. To be quasi-elementary we also require that the computational complexity of $g(x)$ must be less than the computation complexity of enumerating and evaluating the neighbors of solution x . Like the cost function $f(x)$, the auxiliary function $g(x)$ can access all of the components (e.g., the cost matrix) and can compute a “correction” relative to solution x to account for the fact that the landscape is almost, but not quite, elementary.

For those quasi-elementary landscapes that are described in this paper, we can provide additional information about the function $g(x)$. First, computing $g(x)$ has complexity no greater than computing $f(x)$; and when $y \in N(x)$ and $f(y)$ can be computed as a partial update to $f(x)$, then $g(y)$ can also be computed as a partial update to $g(x)$. Second, assume that the set of components is broken into 3 sets: $f(x)$, $f'(x)$ and $\sum_{c \in C} c - f(x) - f'(x)$ where $f'(x)$ identifies (and sums) a set of components relative to x where those components that contribute to $f'(x)$ are sampled at a different rate relative to the other two subsets of components. Then assume that

$$g(x) = p_4 f'(x) - p_2 f'(x)$$

where p_4 is the new sampling rate for the components that contribute to $f'(x)$. Assume the complexity of computing $f'(x)$ is no greater than the complexity of computing $f(x)$. We then obtain the following result.

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) - p_1 f(x) + p_4 f'(x) + p_2 \left(\frac{\bar{f}}{p_3} - f(x) - f'(x) \right) \quad (2)$$

When expressed in this form, we can see that instances of a quasi-elementary landscape can be elementary. If $p_4 = p_2$ for a specific problem size n , the landscape becomes elementary because $f'(x)$ drops out of the calculation. On the other hand if $\bar{f}/p_3 - f(x) = f'(x)$ for a particular problem size, a different special case occurs:

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) - p_1 f(x) + p_4 f'(x) = f(x) - p_1 f(x) + p_4 (\bar{f}/p_3 - f(x)) \quad (3)$$

Examples of both situations occur in this paper within the same domain.

1.1 Two Forms of Quasi-Elementary Landscapes.

Quasi-elementary landscapes can arise in (at least) two distinct ways.

One way quasi-elementary landscapes can occur is if the evaluation function is still made up of a linear combinations of components such as the weights of a cost matrix; furthermore, most of the changes that occur across the entire neighborhood correspond to the wave equation, but a few changes are not captured by the wave equation. In this case, those additional changes in the components must concisely be captured by a function $g(x)$.

The other way that a quasi-elementary landscape can arise is when the quasi-elementary landscape characterizes a partial neighborhood of an otherwise elementary landscape. This is particularly interesting when we are given an elementary landscape where $\alpha > \beta$. Note that $\alpha \neq \beta$ because α/d characterizes components that are drawn for removable from the current solution x . Recall that $C - x$ denotes the set of components not found in the current solution. Thus β/d characterizes the proportion of new components draw from $C - x$. The normalization with respect to d should not cloud the fact that we sample from different size sets when sampling from x and $C - x$; normally, $|C - x| > |x|$, which implies $\alpha > \beta$. Given $\alpha > \beta$ the components in x are sampled at a higher rate than the components in $C - x$ and there must exist “partial” neighborhoods that partially obey the wave equation. Differences between the full neighborhood of an elementary landscape and the partial neighborhood of an elementary landscape can also be captured in a function $g(x)$. Partial neighborhoods of elementary landscapes have been studied before, but they have not been identified as “quasi-elementary” landscapes. The partial neighborhoods of the Traveling Salesman Problem under 2-opt have been presented in detail by Whitley and Ochoa [12]. Partial neighborhoods for Graph Coloring and Min-Cut Graph partitioning have been defined by Whitley and Sutton [13]. Partial neighborhoods as quasi-elementary landscapes will not be addressed in the current paper.

It should be also be noted that some landscapes that are not elementary can nevertheless be expressed as a superposition of a small number of elementary landscapes. For example, a MAX-3SAT landscape under the Hamming-1 neighborhood is not elementary, but it can be re-expressed as the sum of three functions. Let $M(x)$ denote the MAX-3SAT evaluation function for a Boolean string x ; there exists functions f_1, f_2 and f_3 such that the landscapes of f_1, f_2 and f_3 are elementary, and

$$M(x) = f_1(x) + f_2(x) + f_3(x)$$

This makes it not only possible to computing averages over the Hamming-1 neighborhood, it is also possible to compute higher order statistical moments (variance, skew, kurtosis) and it is also possible to compute these statistics in polynomial time over arbitrary Hamming Balls in the landscape, even over regions that are exponentially large [10]. The same method can be applied to all k -bounded pseudo-Boolean functions, including NK-Landscapes [6]. Thus, there is a great deal that we can potentially compute about local search landscapes that is not being utilized by search algorithms.

In theory, we already know that quasi-elementary landscapes can be re-expressed as a superposition of elementary landscapes; the only real question is whether they can be expressed as a superposition of a small fixed number of elementary landscapes. This question is left for future research.

2 Background

We formally define elementary landscapes as follows. Let X be a set of solutions, $f : X \rightarrow \mathbb{R}$ be a fitness function, and $N : X \rightarrow \mathcal{P}(X)$ be a neighborhood operator. Thus, $N(x)$ defines the set of points that are neighbors of solution x . The neighborhood operator can also be represented by its *adjacency matrix*:

$$\mathbf{A}_{xy} = \begin{cases} 1 & \text{if } y \in N(x) \\ 0 & \text{otherwise} \end{cases}$$

Since a discrete function over the set of candidate solutions $g : X \rightarrow \mathbb{R}$ can be characterized as a vector in $\mathbb{R}^{|X|}$, any $|X| \times |X|$ matrix can be used as a linear operator on that function. We will restrict our attention to *regular* neighborhoods, where $|N(x)| = d$, a fixed constant for all $x \in X$. When a neighborhood is regular, the *Laplacian operator* is $\Delta = \mathbf{A} - d\mathbf{I}$ and its influence on the fitness function f is:

$$\Delta f(x) = \sum_{y \in N(x)} (f(y) - f(x))$$

Stadler defines the class of *elementary landscapes* where the objective function f is an eigenfunction of the Laplacian of the graph induced by the neighborhood operator [8]. If function f has been centered to have zero mean, then $-k$ is the eigenvalue of Δf .

$$\Delta f = k(\bar{f} - f) = -k(f)$$

Note that in general f need not have zero mean, and $\Delta f = k(\bar{f} - f)$. For all elementary landscapes it is possible to compute $\bar{f}$, the average evaluation over the entire search space. We can also compute $\text{Avg}(N(x))$, the average evaluation over all of the neighbors of any solution x .

$$\begin{aligned} \text{Avg}(N(x)) &= \frac{1}{d} \sum_{y \in N(x)} f(y) = \frac{1}{d} \left(\sum_{y \in N(x)} f(y) - f(x) \right) + f(x) = \frac{1}{d} \Delta f(x) + f(x) \\ &= f(x) + \frac{k}{d} (\bar{f} - f(x)) \end{aligned}$$

From this equation one can derive many fundamental properties of elementary landscapes by simple algebra. One common property of elementary landscapes is that either

$$f(x) < \text{Avg}(N(x)) < \bar{f} \quad \text{or} \quad f(x) > \text{Avg}(N(x)) > \bar{f}$$

as long as $f(x) \neq \text{Avg}(N(x))$ and $0 < k/d < 1$. An elementary landscape that has the property that $0 < k/d < 1$ is said to be “smooth.” This means that all local minima have an evaluation less than $\bar{f}$ and that all local maxima have an evaluation greater than $\bar{f}$. Furthermore, $f(x) \neq \text{Avg}(N(x))$ unless $f(x) = \text{Avg}(N(x)) = \bar{f}$. This means that all neighborhoods have either improving or disimproving moves, unless the *entire* search space is flat [1].

Barnes et al. [1] introduced the concept of *smooth* elementary landscapes and *rough* elementary landscapes. A rough elementary landscapes has the property that $1 < k/d < 2$. Note that $k/d = 1$ implies $\text{Avg}(N(x)) = \bar{f}$. Most known elementary landscapes are smooth. Rough elementary landscapes have the rather odd behavior that

$$f(x) < \bar{f} < \text{Avg}(N(x)) \quad \text{or} \quad f(x) > \bar{f} > \text{Avg}(N(x))$$

but rough elementary landscapes do appear in interesting contexts, such as in the Walsh decomposition of MAX-3SAT landscapes [10].

3 The Component Based Model of Elementary Landscapes

For the elementary landscapes we have examined (TSP, Graph Coloring, Min-Cut and Max-Cut Graph Partitioning, Basic Frequency Assignment and select pseudo-Boolean functions), the “components” that make up a solution x can be decomposed [3, 11, 14, 13]. Typically, these components are weights in a cost matrix (e.g., distance matrix in TSP). The components of the neighborhood $N(x)$ can be separated into two parts: 1) those weights that contribute to $f(x)$ and 2) those weights that do not contribute to $f(x)$. We exploit this property in identifying elementary landscapes.

Let C denote the set of components used to construct the cost function. We can also view solution x as just a subset of cost components. By a slight abuse of notation, we will let $(C - x)$ refer to the subset of components in C that do not contribute to $f(x)$. When transforming x to some $y \in N(x)$, we subtract a subset of components from x and add new components from $(C - x)$ to the remaining components of x to compute costs of y . If a landscape is elementary, all components in x are uniformly sampled for potential removal from x and all components in $(C - x)$ are uniformly sampled in the set of neighbors denoted by $N(x)$.

The following three ratios, p_1 , p_2 and p_3 , correspond to changes in sampling rates. Let p_1 denote the proportion of components in $f(x)$ that change when a move is made; p_2 is the proportion of components in $(C - x)$ that change when a move is made. And p_3 is the proportion of the total components in C that contribute to the cost function for any randomly chosen solution such that $\bar{f} = p_3 \sum_{c \in C} c$. For p_1 , p_2 and p_3 to be valid, sampling of the cost components must be *uniform*. For example, every weight in the cost matrix appears a uniform (equal) number of times across the space of all possible solutions. Every weight in $f(x)$ appears a uniform number of times in every neighborhood and every weight in $(C - x)$ appears a uniform number of times in every neighborhood. This is the critical insight that explains *why* we can use the

	Edges in x					Edges in $(C - x)$				
	w_{ab}	w_{bc}	w_{cd}	w_{de}	w_{ae}	w_{ac}	w_{ad}	w_{bd}	w_{be}	w_{ce}
$x = \text{ABCDE}$	1	1	1	1	1	0	0	0	0	0
$y_1 = \text{ABEDC}$	1	0	1	1	0	1	0	0	1	0
$y_2 = \text{ABCED}$	1	1	0	1	0	0	1	0	0	1
$y_3 = \text{ABDCE}$	1	0	1	0	1	0	0	1	0	1
$y_4 = \text{ACBDE}$	0	1	0	1	1	1	0	1	0	0
$y_5 = \text{ADCBE}$	0	1	1	0	1	0	1	0	1	0
CHANGE	-2	-2	-2	-2	-2	2	2	2	2	2

Table 1 How edges uniformly move in and out of solution x and the neighborhood $N(x)$.

eigenvalue from the Laplacian to compute neighborhood averages. The sum of the cost components in the set $(C - x)$ can be calculated by

$$(\sum_{c \in C} c) - f(x) = \bar{f}/p_3 - f(x)$$

and therefore we can derive the wave equation from the following observation:

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) - p_1 f(x) + p_2 ((\sum_{c \in C} c) - f(x)) \quad (4)$$

Assume the neighborhood size is d . Both p_1 and p_2 can be expressed relative to d ; p_3 is independent of the neighborhood size.

Theorem 1 Assume p_1 , p_2 and p_3 can be defined over any regular landscape such that the evaluation function can be decomposed into components where

$$p_1 = \alpha/d \quad \text{and} \quad p_2 = \beta/d \quad \text{and} \quad \bar{f} = p_3 \sum_{c \in C} c = \frac{\beta}{\alpha + \beta} \sum_{c \in C} c$$

and p_1, p_2 and p_3 correspond to uniform sampling rates of the components; then the landscape is elementary and

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) + \frac{\alpha + \beta}{d} (\bar{f} - f(x)) \quad \text{where } k = \alpha + \beta$$

A proof follows from simple substitution into equation 2 [13]. An example will illuminate these ideas. Consider the neighborhood for a 5 city TSP using the standard 2-opt neighborhood (see Table 1). The set C contains all of the costs (edges) in the cost (distance) matrix. Let w_{ab} denote an edge between vertices A and B. One can see from this example that the sampling rates are uniform over edges in x and $C - x$; this remains true independent of problem size.

Therefore, for the TSP under 2-opt we observe:

$$p_1 = \frac{2}{n} = \frac{n-3}{n(n-3)/2} \quad \text{and} \quad p_2 = \frac{2}{n(n-3)/2}$$

because 2 edges are changed by 2-opt, and there are n edges in solution x and there are $n(n-1)/2 - n = n(n-3)/2$ edges in the set $C - x$. We next calculate k and the neighborhood average

$$k = \alpha + \beta = n - 3 + 2 = n - 1$$

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) + \frac{n-1}{n(n-3)/2} (\bar{f} - f(x))$$

	Edges in x					Edges in $(C - x)$				
	w_{ab}	w_{bc}	w_{cd}	w_{de}	w_{ae}	w_{ac}	w_{ad}	w_{bd}	w_{be}	w_{ce}
$x = ABCDE$	1	1	1	1	1	0	0	0	0	0
$y_1 = B\bar{A}CDE$	1	0	1	1	0	1	0	0	1	0
$y_2 = BC\bar{A}DE$	0	1	0	1	0	1	1	0	1	0
$y_3 = C\bar{B}DEA$	0	1	0	1	1	1	0	1	0	0
$y_4 = CD\bar{B}EA$	0	0	1	0	1	1	0	1	1	0
$y_5 = D\bar{C}EAB$	1	0	1	0	1	0	0	1	0	1
$y_6 = DE\bar{C}AB$	1	0	0	1	0	1	0	1	0	1
$y_7 = E\bar{D}ABC$	1	1	0	1	0	0	1	0	0	1
$y_8 = EA\bar{D}BC$	0	1	0	0	1	0	1	1	0	1
$y_9 = AE\bar{B}CD$	0	1	1	0	1	0	1	0	1	0
$y_{10} = AB\bar{E}CD$	1	0	1	0	0	0	1	0	1	1
CHANGE	-5	-5	-5	-5	-5	5	5	5	5	5

Table 2 How edges uniformly move in and out of solution x and the neighborhood $N(x)$ under a “shift” operator. The underlined vertex is the vertex being moved. For this problem size, the landscape is elementary, but for larger problem sizes the landscape is not elementary.

4 A Quasi-Elementary Landscape

To illustrate a landscape that is quasi-elementary, we will examine the “shift” operator as applied to the Traveling Salesman Problem. Assume a permutation representation is used. A shift operator works by deleting one vertex from the permutation, then that vertex is re-inserted at every other possible position in the permutation. When done at every possible position, this yields some duplicate neighbors. These duplicates can be eliminated by 1) shifting the vertex to be deleted to the beginning of the permutation, and 2) doing insertion to the next $n - 3$ possible positions. This will capture all unique neighbors without duplicates. Therefore, the neighborhood size is $n(n - 3)$.

One way to establish that a landscape might be elementary is to enumerate a small example. However, finding that a particular problem size has an elementary landscape for a given operator does not establish that all problem sizes induce elementary landscapes for a particular problem domain. Instances of a specific size can be elementary, while other instances of different size may not be elementary. The “shift” operator indeed displays a landscape that appears to be elementary when $n = 5$ for the TSP; this is possible because $|x| = |C - x| = 5$. This is illustrated in table 3. It also induces an elementary landscape when $n = 4$. However, the landscapes is not elementary for $n = 6$, in part because $|x| \neq |C - x|$. Table 4 shows the neighbors that are induced with $n = 6$.

Let $f'(x)$ denote an auxiliary function to $f(x)$. Clearly, $f(x)$ evaluates solution x . In contrast, $f'(x)$ deletes each vertex in solution x ; each deletion removed 2 edges, and introduces 1 new edge: $f'(x)$ ignores the deleted edges, but counts the cost associated with the new edges. Since n vertices are deleted, $f'(x)$ is the sum of the n new edges. These new edges are exactly those edges that appear with greater frequency in the set $C - x$. We will use the auxiliary function $f'(x)$ to make a correction to Grover’s wave equation.

	Edges in x						Edges in $(C - x)$								
	w_{ab}	w_{bc}	w_{cd}	w_{de}	w_{ef}	w_{fa}	w_{ac}	w_{ad}	w_{ae}	w_{bd}	w_{be}	w_{bf}	w_{ce}	w_{cf}	w_{df}
$x = ABCDEF$	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
$y_1 = \underline{B}ACDEF$	+1	0	1	1	1	0	1	0	0	0	0	1	0	0	0
$y_2 = BC\underline{A}DEF$	0	1	0	1	1	0	1	1	0	0	0	1	0	0	0
$y_3 = BCD\underline{A}EF$	0	1	1	0	1	0	0	1	1	0	0	1	0	0	0
$y_4 = \underline{C}BDEFA$	0	+1	0	1	1	1	1	0	0	1	0	0	0	0	0
$y_5 = CDBEFA$	0	0	1	0	1	1	1	0	0	1	1	0	0	0	0
$y_6 = CDE\underline{B}FA$	0	0	1	1	0	1	1	0	0	0	1	1	0	0	0
$y_7 = DCEFAB$	1	0	+1	0	1	1	0	0	0	1	0	0	1	0	0
$y_8 = DECFAB$	1	0	0	1	0	1	0	0	0	1	0	0	1	1	0
$y_9 = DEF\underline{C}AB$	1	0	0	1	1	0	1	0	0	1	0	0	0	1	0
$y_{10} = \underline{E}DFABC$	1	1	0	+1	0	1	0	0	0	0	0	0	1	0	1
$y_{11} = EFDABC$	1	1	0	0	1	0	0	1	0	0	0	0	1	0	1
$y_{12} = EFA\underline{D}BC$	0	1	0	0	1	1	0	1	0	1	0	0	1	0	0
$y_{13} = \underline{F}EABCD$	1	1	1	0	+1	0	0	0	1	0	0	0	0	0	1
$y_{14} = FAEBCD$	0	1	1	0	0	1	0	0	1	0	1	0	0	0	1
$y_{15} = FAEBCD$	1	0	1	0	0	1	0	0	0	0	1	0	1	0	1
$y_{16} = \underline{A}FBCDE$	0	1	1	1	0	+1	0	0	1	0	0	1	0	0	0
$y_{17} = ABFCDE$	1	0	1	1	0	0	0	0	1	0	0	1	0	1	0
$y_{18} = ABC\underline{F}DE$	1	1	0	1	0	0	0	0	1	0	0	0	0	1	1
CHANGE	-8	-8	-8	-8	-8	-8	6	4	6	6	4	6	6	4	6

Table 3 An illustration of how edges move in and out of solution x and the shift neighborhood $N(x)$. The underlined vertex is the vertex being moved. In this case, edges are uniformly removed from solution x , but these are sampled from the set $C - x$ at two different rates. An entry of +1 means that an edge was removed, and then reintroduced during reinsertion of the vertex.

THEOREM:

The “shift” operator induces a quasi-elementary landscape for the Traveling Salesman Problem. For $n > 3$ the neighborhood average for the “shift” operator is given by:

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) - \frac{3n-10}{n(n-3)}f(x) + \frac{n}{n(n-3)}f'(x) + \frac{4}{n(n-3)}\left(\frac{n-1}{2}\bar{f} - f(x) - f'(x)\right)$$

PROOF:

We first deal with the symmetries that exist in the shift neighborhood for the TSP. Normally, when a vertex is deleted and reinserted, this cuts 3 edges: it cuts the edge to the left and to the right of the deleted vertex, and it cuts the edge where the deleted vertex is reinserted. Because every vertex will be deleted, and it will be inserted into all possible (non-duplicate) positions, in those cases where 3 edges are removed from $f(x)$, we find that symmetry holds. We can group the neighbors into subsets of size $n-3$ where the same vertex is deleted from x , but the deleted vertex is reinserted into all feasible positions. In this subset of $n-3$ neighbors, the edge to the right of the deleted vertex is reinserted once (e.g., if AB represent the two vertices and A is deleted and re-inserted after B, the edge ab is recovered). This happens 1 time for every edge in solution x across all neighbors.

We can calculate the rate at which edges are removed from x by assuming the symmetry case as a baseline, then correct for the 1 neighbor where a specific edge from x is reinserted. This mean that the rate with which edges are removed from x is given by

$$-\frac{3}{n}f(x) + \frac{1}{n(n-3)}f(x) = \frac{-3n+10}{n(n-3)}f(x).$$

We next deal with the sampling of edges from the set $C - x$. For the TSP, the calculation of $\bar{f}$ is operator independent, therefore:

$$((\sum_{c \in C} c) - f(x)) = (\frac{\beta + \alpha}{\beta} \bar{f}) - f(x) = (\frac{n-1}{2} \bar{f}) - f(x)$$

Next, we observe that the edges in the set $C - x$ are sampled in two different ways. All of the edges are sampled in a symmetric fashion, but a subset of edges receive additional samples.

The symmetric case can be described as follows. New edges are created by inserting the deleted vertex in a new position. Since the insertion occurs in every position that does not produce a redundant neighbor, this case is symmetric. This can happen in only 4 ways: vertex A has been deleted and it is inserted before and after Z; or vertex Z is deleted and it is inserted before and after vertex A. Thus, the symmetric sampling rate over the entire neighborhood is $4/(n(n-3))$. The non-symmetric case derives from the fact that when a vertex is deleted, the deletion also creates a new edge. Furthermore, the *same* edge is created $n-3$ times for each of the $n-3$ cases where the *same* vertex is deleted. However, one of these $n-3$ cases is also one of the symmetric cases previously counted. Removing this one case, there remains $n-4$ cases where the same edge is created when the same vertex is deleted.

The set of edges that are sampled an additional $n-4$ times can be found starting the with current solution x and then deleting each vertex in x to create a permutation (circuit) of $n-1$ vertices. As each vertex is deleted, one new edge is created: thus, there are n new edges that are created. Let $f'(x)$ be the sum of the n new edges that are created by deleting each vertex one at a time. Each of the new introduced edges are from the set $C - x$.

Note that $|C - x| = n(n-2)/2 - n = n(n-3)/2$. Also note that each of the n edge that contributes to $f'(x)$ appears $n-4$ times across the entire neighborhood. Therefore, the symmetric and non-symmetric sampling from $C - x$ is given by

$$\frac{2}{n(n-3)/2} (\frac{n-1}{2} \bar{f} - f(x)) + \frac{n-4}{n(n-3)} f'(x)$$

Using the sample rate from x and $C - x$ yields the combined effect. Note that the operator is only well-defined when $n > 3$. Therefore:

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) - \frac{3n-10}{n(n-3)} f(x) + \frac{4}{n(n-3)} (\frac{n-1}{2} \bar{f} - f(x)) + \frac{n-4}{n(n-3)} f'(x)$$

This can be rearranged to yield:

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) - \frac{3n-10}{n(n-3)} f(x) + \frac{n}{n(n-3)} f'(x) + \frac{4}{n(n-3)} (\frac{n-1}{2} \bar{f} - f(x)) - f(x)$$

While computing $f'(x)$ with no prior knowledge requires $O(N)$ time, if we currently know the evaluation of $f(x)$ and $f'(x)$ and we move to a point y such that $y \in N(x)$, then both $f(y)$ and $f'(y)$ can be computed as a partial update to $f(x)$ and $f'(x)$ respectively, and both partial updates can be computed in constant time. Thus, the resulting landscape is quasi-elementary. $\square$

We find that the landscape behaves differently when $n = 4$ and $n = 5$, but for different reasons. When $n = 5$ we find that $\frac{n-1}{2}\bar{f} - f(x) - f'(x) = 0$ and $3n - 10/n(n-3) = n/n(n-3) = 1/2$ and

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) - \frac{1}{2}f(x) + \frac{1}{2}f'(x) = \bar{f}$$

When $n = 4$ we find that $\frac{n}{n(n-3)} = \frac{4}{n(n-3)}$, $f'(x)$ drops out, and $\frac{3n-10+4}{n(n-3)} = \frac{4}{n(n-3)} \frac{n-1}{2} = 1/2$ and we obtain:

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) + \frac{1}{2}(\bar{f} - f(x))$$

It would seem that these are the only two special cases which occur, however.

4.1 The Maximal Clique Problem

A landscape would not be interesting if it only applied to a single problem. While we can show that other quasi-elementary landscapes exist in problems that are similar to the TSP, we instead look next at a landscape for the maximal clique problem.

Katayama et al. [5] propose a variable neighborhood “k-opt” move operator for the maximal clique problem. (This is not the k-opt operator used for the Traveling Salesman Problem.) Katayama et al. report that this operator is competitive with other heuristic search methods for generating solutions to the maximal clique problem. While the operator has been applied to the maximal clique problem, it can also be used to search for subgraphs with maximal density independent of whether the subgraph is a clique or not.

The operator acts as a kind of graph partitioning heuristic. The operator breaks a graph G with vertices V and edges E into two subgraphs, one with a set of vertices denoted by Z , and the other with the remaining vertices, $V - Z$. Under the “k-opt” move operator, k vertices in Z are removed and k vertices from $V - Z$ are added. We will say that an edge $e_{i,j}$ belongs to Z if vertices v_i and v_j are in the subgraph Z . By bringing in all subsets of k vertices that are not currently in Z into subset Z , all edges that do not belong to Z come into Z . This is one characteristic of an elementary landscape.

Another necessary pre-condition for there to be an elementary landscape (or quasi-elementary landscape) is that $\bar{f}$ must be computable as a uniform sample over the cost components. Assume we are attempting to maximize the number of edges in subgraph $Z \subset V$. Let f count the number of edges in Z . Let $r = |Z|$ and let $q = |V - Z|$. Using counting arguments, one can prove that over all possible assignments of vertices to Z is given by:

$$\bar{f} = \frac{r(r-1)}{q(q-1)} \sum_{c \in C} = \frac{r(r-1)}{q(q-1)} |E|$$

To have an elementary landscape, we need to divide “components” into those that are in solution x , and those that are not in solution x . And each set needs to be sampled at a uniform frequency corresponding to p_1 or p_2 . But when counting (and maximizing) the number of edges in Z and the number of edges not in Z , we can classify edges into 3 different types that are uniformly sampled.

Type 1: Edges that belong to Z and contribute to $f(x)$.

Type 2: Edges in $(C - x)$ and connect two vertices not in Z .

Type 3: Edges in $(C - x)$ and connect a vertex in Z to a vertex in $(V - Z)$.

If the sampling rate of the Type 2 edges happens to be the same as the sampling rate of the Type 3 edges, then the sampling rate for all the edges in $(C - x)$ will be uniform and the landscape will be elementary. One can construct examples where this does happen. But, this does not happen for all problem sizes.

Therefore we can compute p_1 and p_3 , but there are two sampling rates instead of one for the edges normally accounted for by p_2 . Nevertheless, this requires only a minor adjustment. We will denote the two different sample rates by p'_2 and p_4 .

In the current paper we will only look at 2-opt (again, not the TSP 2-opt): we will assume that 2 vertices are swapped between Z and $V - Z$ and that all possible ways of selecting 2 vertices from Z and 2 vertices from $V - Z$ will be considered. Since this is done uniformly in a symmetric fashion, we can express our calculation in terms of the probability of an edge moving out of Z when any randomly chosen neighbor is sampled. These same results can (and have) been generated using combinatorial counting arguments that describe all the ways that neighbors can be generated.

THEOREM:

The “2-opt” operator for maximizing the density of edges in a subgraph Z of graph G induces a quasi-elementary landscape. Let $r = |Z|$ and let $q = |V| - |Z|$. For $r > 2$ and $q > 2$ the neighborhood average for the “2-opt” operator is given by:

$$\text{Avg}_{y \in N(x)}(f(y)) = f(x) - \frac{4r-6}{r(r-1)}f(x) + \frac{2(r-2)}{rq}f'(x) + \frac{2}{q(q-1)}\left(\frac{q(q-1)}{r(r-1)}\bar{f} - f(x) - f'(x)\right)$$

PROOF:

We have already computed $p_3 = \frac{r(r-1)}{q(q-1)}$.

We next compute p_1 , the probability that edge $e_{i,j}$ moves out in Z . If two vertices are randomly drawn from Z for exchange, the probability of an edge that is currently in Z remaining in Z is given by

$$\frac{r-2}{r} * \frac{r-3}{r-1} = \frac{(r-2)(r-3)}{r(r-1)}$$

Therefore

$$p_1 = 1 - \frac{(r-2)(r-3)}{r(r-1)} = \frac{4r-6}{r(r-1)}$$

We next compute p'_2 . We define p'_2 to be the probability that an edge “contained” within $V - Z$ moves to Z .

$$p'_2 = \frac{2}{q} * \frac{1}{q-1} = \frac{2}{q(q-1)}$$

Finally, we will compute p_4 . Consider an edge $e_{i,j}$ such that $v_i \in Z$ and $v_j \in V - Z$. Then p_4 corresponds to the probability that v_i stays in Z and v_j moves to Z when any random neighbor is considered.

$$p_4 = \frac{(r-2)}{r} * \frac{2}{q} = \frac{2(r-2)}{rq}$$

We now need 3 counts of edges. 1) $f(x)$ is the number of edges that appear in Z for solution x . 2) $f'(x)$ is the number of edges with one vertex in Z and one vertex in $V - Z$ for solution x . 3) $\sum_{c \in C} c$ is the total number of edges, which can be computed relative to $\bar{f}$.

Putting these together we obtain.

$$AVG = f(x) - p_1 f(x) + p_4 f'(x) + p_2' (\sum_{c \in C} c) - f(x) - f'(x)$$

And by substitution:

$$AVG = f(x) - \frac{4r-6}{r(r-1)} f(x) + \frac{2(r-2)}{rq} f'(x) + \frac{2}{q(q-1)} (\frac{q(q-1)}{r(r-1)} \bar{f} - f(x) - f'(x))$$

Computing both $f(x)$ and $f'(x)$ can be done by enumerating the vertices in Z and checking the edges that are incident on vertices in Z . Therefore $f'(x)$ can be computed as a side-effect of computing $f(x)$. And for $y \in N(x)$, both $f(y)$ and $f'(y)$ can be computed as a partial update to $f(x)$ and $f'(x)$ respectively. Therefore computing $f'(x)$ has complexity less than or equal to computing $f(x)$. $\square$

These results could be expressed with respect to the neighborhood size. The total size of the neighborhood is given by

$$d = \binom{r}{2} \binom{q}{2} = r(r-1)q(q-1)/2.$$

This general form may describe other types of quasi-elementary landscapes, particularly for problems that involve partitioning graphs where edges can be “in” and “out” and “between.”

It is again possible that $p_4 = p_2'$ and the landscape becomes elementary. This happens when $r = q = 3$. But it does not happen in general.

5 Conclusions

There are several NP-hard problems that display *elementary landscapes* when the problem is posed as a local search problems with one or more different local search move operators. The “component model” shows that many cost functions that display elementary landscapes use a cost matrix, and the weight in the cost matrix can be partitioned into those that contribute to solution x and those that do not contribution to solution x .

In this paper we have shown that quasi-elementary landscapes exists for problems when the cost matrix can be divided into 3 subsets: 1) those cost that contribution to solution x , 2) those costs that do not contribute to x , and are samples at rate p_2 in $N(x)$, and 3) those costs that do not contribute to x , and are sampled at rate p_4 in $N(x)$. This might be generalized to cases where there are more than 2 sampling rates for subsets of $C - x$.

It might also be the case that all quasi-elementary landscapes can be re-expressed as a superposition of a small bounded number of elementary landscapes [2]. Indeed this seems likely. Future research will explore whether a general form exists that can be used to map quasi-elementary landscapes onto a superposition of a small number of elementary landscapes.

Acknowledgements This research was also sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-08-1-0422. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

1. J. W. Barnes, B. Dimova, and S. P. Dokov. The theory of elementary landscapes. *Applied Mathematics Letters*, 16(3):337–343, April 2003.
2. F. Chicano, D. Whitley, and E. Alba. A methodology to find the elementary landscape decomposition of combinatorial optimization problems. *Evolutionary Computation*, 2011.
3. F. Chicano, D. Whitley, E. Alba, and F. Luna. Elementary landscape decomposition of the frequency assignment problem. *Theoretical Computer Science*, 2011.
4. Lov K. Grover. Local search and the local structure of NP-complete problems. *Operations Research Letters*, 12:235–243, 1992.
5. K. Katayama, A. Hamamoto, and H. Harihisa. An effective local search for the maximum clique problem. *Information Processing Letters*, 95(5):503–511, 2005.
6. Stuart Kauffman and Simon Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128:11–45, 1987.
7. Peter F. Stadler. Toward a theory of landscapes. In R. López-Peña, R. Capovilla, R. García-Pelayo, H. Waelbroeck, and F. Zertruche, editors, *Complex Systems and Binary Networks*, pages 77–163. Springer Verlag, 1995.
8. Peter F. Stadler. Landscapes and their correlation functions. *Journal of Mathematical Chemistry*, 20:1–45, 1996.
9. A. Sutton, D. Whitley, and A. Howe. A polynomial time computation of the exact correlation structure of k-satisfiability landscapes. In *Genetic and Evolutionary Computation Conference*, pages 381–388. ACM, July 2009.
10. A. Sutton, D. Whitley, and A. Howe. Computing the moments of k-bounded pseudo-boolean functions over hamming spheres of arbitrary radius in polynomial time. *Theoretical Computer Science*, 2011.
11. D. Whitley, F. Chicano, , E. Alba, and F. Luna. Elementary landscapes of frequency assignment problems. In *GECCO-10*. ACM Press, 2010.
12. Darrell Whitley and Gabriela Ochoa. Partial neighborhoods of the traveling salesman problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Dublin, Ireland, July 2011.
13. Darrell Whitley and Andrew M. Sutton. Partial neighborhoods of elementary landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Montreal, Canada, July 2009.
14. L. Darrell Whitley, Andrew M. Sutton, and Adele E. Howe. Understanding elementary landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Atlanta, GA, July 2008.

Papers

An Evolutionary Algorithm for the large Size One-Commodity Pickup and Delivery Travelling Salesman Problems

Istvan Borgulya

Abstract In this paper we present an evolutionary algorithm (EA) for the one-commodity pickup and delivery travelling salesman problem (1-PDTSP). This EA is a hybrid steady-state algorithm that uses truncation selection, uses only mutation for the generation of the descendants and it improves the solutions with three local search procedures. The mutation is based on “the EVL method”, that is a memory-based method. The algorithm was tested on large size benchmark problems; the quality of the results is good, the algorithm outperforms the previous heuristics in 84% of the large size test cases.

1 Introduction

In this problem goods are transported between pickup and delivery locations with a single vehicle with a limited capacity. The vehicle has to collect all commodities and every unit picked up can be used to satisfy every delivery customer’s demand. The route of the vehicle should start and end at a depot and the vehicle has to serve all pickup and delivery customers exactly once. The goal of the problem is to find a minimum cost route for the vehicle that serves all customers without violating the vehicle capacity.

There are many pickup-and-delivery routing problems described in the literature. Three examples for the one-commodity pickup-and-delivery travelling salesman problem are the milk transfer from cow farms to private residences with no special requirements on sources and destinations; money moved between branches of a bank [6]; transferring a certain medicament from some hospitals to other hospitals, which are in short of this medicament [9].

By the classification of Parragh et al. [10] this problem is a “Vehicle Routing Problem with Pickups and Deliveries”, with unpaired pickup and delivery points and with a single vehicle. In the literature this problem is called various ways e.g. capacitated travelling salesman problem with pickups and deliveries [1], one-commodity pickup and delivery travelling salesman problem [5], and travelling salesman problem with pickup and delivery [6].

The 1-PDTSP is NP-hard since it is equivalent with the TSP when the vehicle capacity is large enough. Up to 60 customers we can use exact algorithms, e.g. the branch-and-cut algorithm [6]. For larger problems we can use various heuristic methods. In [7] Hernandez et al. developed two heuristic algorithms. The first one is a greedy algorithm with a simple local search procedure; the second one is an incomplete optimization algorithm to find the solution in a restricted search space. In [8] a hybrid heuristic algorithm was proposed, the GRASP/VND (Greedy Randomized Adaptive Search procedure/ Variable Neighbourhood Descent). In this two phase method the GRASP is a construction phase and the VND is a local search phase to improve the solutions. In [13] a GA was introduced. This algorithm uses first a constructing heuristic to generate feasible initial population, after it uses the GA with a new

Istvan Borgulya
University of Pecs, Faculty of Business and
Economics, Hungary
borgulya@ktk.pte.hu

pheromone-based crossover operator and with a 2-opt local search. In [9] a new VNS /SA (Variable Neighbourhood Search / Simulated Annealing) method is presented. The VNS changes the neighbourhood size adaptively and an SA acceptance criterion is used to allow the algorithm to escape local optima. It uses also a 2-opt local search to improve the solutions.

As the previous examples show the hybrid metaheuristics can solve the hard combinatorial problems well. There are hybrid EAs, too (e.g. in [13]), and in general there are a lot of combinations of exact algorithms or metaheuristics with EAs [11], [12]. The very frequently used hybrid EAs improves the candidate solutions with local search procedures. For example, in memetic algorithms, various kinds of local search are embedded in an EA locally improving candidate solutions obtained from variation operators.

We wish to enrich the field of these heuristic methods with a new hybrid EA. This is a steady-state memetic algorithm that uses truncation selection, uses only mutation for the generation of the descendants and improves the solutions with three local search procedures. The mutation is based on “the EVL method”, that is a memory-based method [3], [4]. In our EA the individuals of the population are routes and a route is a permutation of the customers; every position in the permutation is an index of a customer.

New techniques in our algorithm:

- we adapted the EVL method to estimate the probability of the mutation. This EVL method uses the worst individuals of the population in terms of the fitness function. It memorizes the relative frequencies of the permutation values on the different positions by the worst individuals, and stores this information in a matrix.
Now we also memorize and store in another matrix how severely these values violated the capacity condition of the vehicle. With the help of this extra information we can improve the estimated probability of the mutation.
- we use a new local search procedure to improve a solution and we apply a combination of different local search procedures in the algorithm.

The motivation of our research was the following: to show that we can achieve good results for the 1-PDTSP with the help of the EVL method and simple local search procedures without the use of sophisticated recombination or mutation operators.

The paper is organized in the following way: section 2 describes the problem formulation; section 3 shows the new algorithm. Section 4 describes some details of the algorithm: the applied local search methods, the EVL method with the mutation operator, and the parameter selection. Section 5 shows our computational experience with the new EA and compares our algorithm to other heuristics. Section 6 concludes.

2 Problem formulation

Based on [13], [7] the 1-PDTSP can be defined as follows. Let $G=(V,E)$ represent a complete, undirected graph, where $V=\{1,2,\dots, n\}$ is the vertex set and E is the edge set. Vertex 1 represents the depot; the other vertices represent the customers. A non-zero demand q_i associated with each customer i is given. If $q_i>0$ then v_i is a pickup customer, otherwise is a delivery customer. The depot can be considered a customer by

$$q_1 = -\sum_{i=2}^n q_i.$$

For each edge $(i,j) \in E$ a non-negative cost (or distance) d_{ij} is given. A vehicle with capacity Q must visit each customer exactly once and its capacity must not be exceeded. The 1-PDTSP should determine a route starting and ending at the depot and having minimum cost (length):

$$f = d_{n1} + \sum_{i=2}^n d_{i-1,i}.$$

3 The new algorithm

Let us denote the new EA OCPD (One-Commodity Pickup and Delivery). It is a hybrid EA that improves the new solutions using local search procedures. It uses a 2-stage algorithm structure where both stages are steady-state hybrid EAs. The first stage is a quick “preparatory” stage which first generates $d=\min(t/2, n)$ feasible individuals (where t is the size of the population) and later randomly improves the quality of the initial population.

In the second stage in every generation the algorithm selects a parent by truncation selection and its copy will be the descendant. This stage uses a mutation based on the EVL method and applies three local search procedures to improve the descendant. As additional operation, the algorithm updates the various matrices of the EVL method (called ECMs matrices) and to speed up the convergence it uses the *Delete*, *Filter* and *Restart* procedures. The main steps of OCPD are shown in Algorithm 1.

The parameters of the algorithm:

t - the size of the population,

itt - the number of the generations in the first stage.

kn - the algorithm is controlled in every kn th generation.

$itend$ - the maximal number of generations.

$t\%$ - parameter of truncation selection.

$g\%$ - parameter of the condition of the *Restart* procedure.

$r\%$ - parameter of the *Restart* procedure.

$d\%$ - parameter of the *Delete* procedure.

Algorithm 1. The OCPD algorithm

1. **Procedure** OCPD($t, itt, kn, itend, t\%, g\%, r\%, d\%$)
 2. Generate t individuals. Initialize values of ECMs. $it=0$ /* First stage
 3. **Do** itt times
 4. $it = it + 1$. Generate a descendant randomly. Local search. Reinsertion.
 5. In every kn th generation: Update ECMs. Filter, Delete.
 6. **od.**
 - /* Second stage
 7. **Repeat**
 8. $it = it + 1$. Truncation selection, mutation, local search, reinsertion.
 9. In every kn th generation: Update ECMs. Filter, Delete, Restart.
 10. **until** $it > itend$
 11. **end**
-

The operations and the characteristics are the following:

Individuals. An individual is a permutation of the vertices. Let the depot be at the last position for every individual.

Initial population. In case of the 1-PDTSP to find the first feasible individuals is an important and difficult task. For this we use the construction heuristic of [13], which helps to find a feasible solution with a high probability (named FSC: Feasible Solution Construction heuristic). So the $d=\min(t/2, n)$ first individuals are generated by the FSC construction heuristic and they are improved by the local search procedures. The other individuals of the population are generated from the first individuals with a few random swaps.

Constraint. The algorithm controls the capacity violation by every customer. If the vehicle should exceed the capacity, or it cannot give enough goods for a customer we say the vehicle violated the constraint. Let $D(x)$ be the measure of the violation of the constraint

at the individual (or route) x . If the first violation happened after the i th customer of the route x , then $D(x)=n-i$ (number of remaining customers after the first violation). If x is a feasible solution then $D(x)=0$.

Fitness function. The fitness function is equivalent to the cost function f , but additionally we include $D(x)$ in calculating the fitness. Based on the cost function and the measure D we can compare individuals: individual x is better than individual z if $D(x)<D(z)$. In case $D(x)=D(z)$ the individual x is better if $f(x)<f(z)$.

Selection operator. In the first stage the descendants are randomly generated (there is no selection). In the second stage the algorithm selects only one parent by truncation selection. In this selection only the best $t\%$ of the population is considered as a potential parent.

Mutation operator. The algorithm uses with 0.5 probabilities a swap based on the EVL method (details in section 4.), otherwise uses with 0.5 probabilities a random swap or insert move.

Local-search. The algorithm uses a sequence of three local search procedures one after the other (details in section 4.).

Filter. This procedure filters out and deletes the weak individuals which are members of the neighbourhood of a better individual (based on the Hamming distance, e.g. $d^H(x, x')<2$). In the next generations with the *reinsertion* the deleted individuals will be replaced with new ones (also after the *Delete* and *Restart* procedures).

Restart. If the fittest solution didn't change in the last $g\%$ generations, it deletes the weakest solutions ($r\%$ proportion of the population).

Delete. This procedure deletes the weakest solutions ($d\%$ proportion of the population where $d\%*t<kn$).

Reinsertion. With the help of the reinsertion we want to separate the individuals into different groups based on their convergence to the global or local optima. So we use a special version of the crowding technique that compares the descendant and the former solutions (or the parent) based on the similarity, and selects one of them [2].

In the first stage the descendant may replace the most similar one of the former solutions (based on the Hamming distance between the individuals) if the descendant is better. In the second stage the algorithm selects between the parent and the descendant if the population has t individuals. If the number of individuals is less than t (after *Restart*, *Filter* or *Delete* procedures) the new descendant is unconditionally inserted to the population until the population size is completed.

Stopping criterion. The algorithm is terminated if a pre-determined number of iterations has been reached (*itend* parameter).

4 Details of the implementation

4.1 Local search

In our OCPD algorithm one of the most important operations is the local search. To find appropriate search algorithms we applied different local search procedures in the algorithm. So we tried some 2-opt local search versions, where the move in the procedure was swap, insert, or inversion. We developed a new version of the FSC procedure too, as a local search. This procedure (named IFSC: improving FSC) tries to improve a solution based on the FSC. IFSC begins its improving modification only after the n/s th position of the individual (based on experience $s=4$ is an appropriate value), the earlier permutation values do not change in the individual (Algorithm 2).

We tested the OCPD algorithm with the different local search procedures and with some combinations of the local search procedures. For the test we chose four large problems from the instances created by [6] (the n100q10E, n100q10F, n100q10G and n100q10H problems; notice: n100E, n100F, n100G, n100H). All test problems were run 10 times with the same time

limit. We computed the run statistics of the various versions of our OCPD algorithm (applying various local search procedures) for each test problem. Based on the statistics we concluded the following:

- the 2-opt local search is the best with the inversion move that reverses the sequence of the permutation between two positions (named INV),
- we can use procedures one after the other. We can organize the sequence e.g. INV + IFSC and INV+ IFSC+INV.

Algorithm 2. The IFSC algorithm

1. Let $i=n/s$, if the first n/s customers do not violate the capacity constraint.
If there is a z th customer ($z < n/s$) who first violates the capacity constraint than let $i=z$.
 2. Let T be the first part of the individual with the first i customers.
 3. $i=i+1$
 4. If $i = n$, where n is the total number of customers including the depot, then add the depot into T at the i th position and STOP.
 5. Search all the customers (without the depot) that have not appeared in T . If there are one or more customers who can be appended to T at the i th position without violating the capacity constraint, then select the customer the nearest to $i-1$, otherwise select a random customer, appended the selected to T at the i th position and go to 3.
-

The best found solution (*Best*), the average solution (*Avg*) and the standard deviation of the solutions (*SD*) are summarised in the table 1 at the best local search versions. We can see the results of OCPD with the different local searches as INV, INV+IFSC and INV+IFSC+INV. Based on the average results of the four test problems we concluded: OCPD with the INV+IFSC+INV procedures is the best algorithm version, so we use this local search sequence in the algorithm.

Table 1. Test results with different local search procedures ($Q=10$)

name	OCPD with								
	INV			INV+IFSC			INV+IFSC+INV		
	Best	Avg	SD	Best	Avg	SD	Best	Avg	SD
n100E	11403	11418.0	21.7	11408	11410.6	3.0	11403	11421.8	17.2
n100F	11708	11819.6	139.4	11721	11829.6	109.5	11615	11740.0	37.1
n100G	12005	12064.3	101.0	11966	12038.3	65.0	11966	12050.5	74.1
n100H	12740	12856.3	116.5	12814	12909.6	85.5	12740	12769.0	46.2
Aver.	11964	12039.5	94.6	11977.2	12047.0	65.7	11931	11995.3	43.6

4.2 The EVL and the mutation operator

The principle of the EVL is the following [3]. Let us consider a generic EA, and suppose that the individual has l variables and each variable of the individual can have k discrete values. Let us notice that ECM (explicit collective memory) is a $k \times l$ matrix that stores and learns the relative frequencies of the different values of the variables. This matrix is updated through the evolution procedure using a few of the worst performing individuals.

Let ECM_{ij}^{gen} be the collected relative frequency of the i th values on the j th position (variable) until the gen th generation. We can update the elements of the ECM matrix

$$ECM_{ij}^{gen+1} = (1-\alpha)ECM_{ij}^{gen} + \alpha \Delta ECM_{ij} \text{ (e.g. } \alpha = 0.2)$$

where ΔECM_{ij} is the relative frequency of the i th value on the position j based on the worse individuals of the gen th generation and α denotes some relaxation factor. For the probability of mutating the j th variable in individual X we can use the formula

$$pr_j = 1 - \left| \frac{ECM_{x_j j}^{gen}}{\sum_{k=1}^n ECM_{kj}^{gen}} - a_j \right|$$

where B is one of the best individuals and if $X_j = B_j$ then $a_j = 1$ else $a_j = 0$.

Mutation for 1-PDTSP

The generated ECM is based on the worst individuals of the population. The ECM for 1-PDTSP is an $n \times n$ matrix. Every customer has a row and every position of the individual has a column in the matrix. In every kn th generation the ECM is updated by using the weakest individuals. In this updating procedure we use 20% of the population.

The mutation in 1-PDTSP is based on the EVL method. Let X be a descendant. The mutation operator first chooses a random j position of X (in the permutation). Next it chooses another customer for this position. It chooses the value with the highest pr_j probability; it searches the position of the value and swaps the values of the positions. (The algorithm uses one swap during the mutation.)

We can reach a faster convergence if we apply two versions of the ECM: a long term memory (ECML) and a short term memory (ECMS) [4]. The ECML is updated only during the first 2000 generations and later it does not change. The ECMS is updated continuously but after every 300 generations we delete the value of the ECMS, and we begin the update of ECMS with an empty matrix. Using the two memories we have to compute the pr_j probability in another way. First we have to compute two probabilities for the position j based on the ECML (pl_j) and based on the ECMS (ps_j). Next we compute the pr_j probability with the formula $pr_j = pl_j * ps_j$. Using the new probability (and memories) the quality of the results is better in comparison with the earlier single ECM version.

We can further improve the estimated probability if we use the EVL method for the $D(X)$ measure too. An ECMD matrix like the ECM can store and learn the average of the $D(X)/n$ values of the individuals for each customer and position. Based on the ECMD we can compute a new pd_j probability in a similar way and for the mutation we use a new probability: the $pr_j = pl_j * ps_j * pd_j$. (In parallel with the ECMS and ECML update we can update the ECMD matrix with the same individual too).

We use a problem specific technique at the mutation step, too. During the search of the most appropriate customer for the j th position we examine only the set of the first r nearest customers of the $(j-1)$ th customer, where r is 30 for test cases of 100 customers and 60-70 for 200-500. From this set we select and examine the customers in order of increasing cost one after the other with a probability of 0.7, or select random nearest customers with a probability of 0.3; choosing the first pr_j maximum in case of multiple equal maxima.

During the test we observed that the use of the ECMD improved average results by 0.3 % and we reach more new best known solutions than without the use of the ECMD.

4.3 Parameter selection

Studying some of the more complex problems of the benchmark set we analyzed the process of the convergence to find how the parameter values were affecting the convergence, the finding of the global optimum and the speed of the calculation. So we analyzed the population size (t parameter), the frequency of checks (kn parameter), the generation in the first stage (itt parameter), the parameters of the *Delete* ($d\%$) and *Restart* procedures ($r\%$ and

$g\%$), and of the truncation selection ($t\%$). Summarizing the results of the analysis $t = 90$, $itt = 50$, $kn = 10$, $d\% = 0.1$, $r\% = 0.7$, $g\% = 300$ and $t\% = 0.1$. The maximal number of the generations was 30000 (and the maximal number of the generated individuals, too). This maximal number was chosen based on the publication [13] where the maximal number of the generated individuals was 30000; and that gives a good basis for the comparison of the methods.

5 Experimental results

5.1 Test problems

We tested our algorithm with the instances created by [6] (The data set is available from the web site <http://webpages.ull.es/users/hhperez/PDsite>). In the data set there are small instances with n in $\{20, 30, 40, 50, 60\}$ and large instances with n in $\{100, 200, 300, 400, 500\}$. The vehicle capacity Q takes values in $\{10, 15, 20, 25, 30, 35, 40, 45, 50, 1000\}$. In the data set there are 10-10 instances for each (n, Q) pair.

5.2 Computational experiences

In this paper we tested the large instances with the smallest capacity ($Q = 10$) because they are the most difficult problems. The algorithm was run 10 times on each test case from 100-300 customers and 5 times on test case from 400-500 customers. The OCPD algorithm was implemented in Visual C++ and ran on Intel Core Duo CPU 2.2 GHz with 2 GB RAM under Windows Vista Business.

Table 2 shows the results of OCPD. In the table we give the problem name that gives the n value too (*name*), the best known solution (*BKS*) based on [9], the best found solution (*Best*), the average solution (*Avg*) and the standard deviation of the solutions (*SD*).

Table 2 shows that OCPD was able to improve best known results in 90% of the 100 test cases, 100% of the 200 test cases, 70% of the 300 test cases, 70% of the 400 test cases and 90% of the 500 test cases (these are highlighted with bold characters in table 2). The average of the best found solutions is 0.4% better than the average of the earlier best known solutions.

5.3 Comparative results

To compare the results of OCPD to other methods we chose two heuristic methods, which had the best published results in the literature. These are the genetic algorithm from [13] (notice: GA) and the hybrid VNS/SA from [9] (notice: VNS/SA). The GA was run on a Pentium M 1.6 GHz PC with 256 MB RAM, but the VNS/SA was run on multiple different computers so the authors could not give an “average” characteristic of the computers.

Table 3 shows the comparative results, the average of the best found and the average solutions by every method and by every group by n ($Q=10$). Some data are missing (Not available: N/A).

We can conclude that the results of OCPD are of better quality than VNS/SA and GA in every group of the test problems (these are highlighted with bold characters in table 3). In the case of GA, OCPD improved the best found solutions by 1.9% and improved the average solutions by 2.69%. In the case of VNS/SA, OCPD improved the best found solutions by 0.49%, and in the 100, 200 and 300 test cases it improved by 1.12% the average solutions.

Table 4 shows the average running times until the best found solutions by the different methods. The comparison of running times is encumbered by the use of various programming languages, operating systems and computers. But in general we can say that OCPD is faster than VNS/SA based on the two known data of VNS/SA. The GA is the fastest with the same maximum number of individuals. But its results are weaker than the OCPD results especially among the 500 test cases, where the best results of GA are 3.23% weaker than the ones of

OCPD. So we can conclude that the running time of OCPD is large, but it is generally able to find better solutions than the GA or VNS/SA.

5.4 Computational experiences with other test problems

We developed and tested the OCPD with large instances. At end we tried its efficiency by small instances. We used the OCPD by these test problems with the same parameter values than by the large problems. All test cases were run 5 times.

Table 2. OCPD results on 100-500 customers ($Q=10$)

Name	BKS	Best	Avg	SD	Name	BKS	Best	Avg	SD
n100A	11741	11705	11721.0	20.5	n300F	24042	23861	24246.5	309.6
n100B	13066	13059	13095.6	36.4	n300G	23683	23537	23837.7	194.5
n100C	13893	13893	13912.0	26.0	n300H	21555	21523	21832.3	218.8
n100D	14253	14245	14372.5	175.3	n300I	23871	23870	24149.8	222.6
n100E	11411	11403	11420.5	17.2	n300J	22503	22188	22641.2	256.1
n100F	11644	11615	11645.8	37.1	n400A	30657	30418	30692.3	210.5
n100G	12025	11966	12033.6	74.1	n400B	24248	24167	24479.8	211.5
n100H	12818	12740	12768.3	46.2	n400C	27853	27988	28110.3	140.1
n100I	14025	13909	13966.0	40.1	n400D	23750	24055	24105.3	69.4
n100J	13297	13280	13322.6	44.9	n400E	24798	24702	25027.8	241.9
n200A	17686	17501	17601.3	104.7	n400F	26625	26544	26587.0	54.8
n200B	17608	17610	17658.6	41.3	n400G	23891	23684	23805.0	87.8
n200C	16466	16461	16524.2	50.5	n400H	25348	24684	25175.8	328.0
n200D	21228	21153	21223.7	70.7	n400I	28262	28586	28719.0	157.5
n200E	19220	19211	19301.8	83.2	n400J	24847	25343	25694.4	229.1
n200F	21627	21394	21652.0	182.3	n500A	27904	27706	27840.3	230.9
n100G	17361	17359	17434.7	54.1	n500B	26612	26202	26384.0	166.0
n200H	20953	20942	21062.7	111.1	n500C	30247	29948	30332.3	371.3
n200I	18020	17862	18032.0	123.3	n500D	29875	29647	29866.3	239.0
n200J	18970	18945	19051.6	108.7	n500E	29978	29523	29898.8	300.3
n300A	22940	22929	23131.8	147.4	n500F	28527	27992	28261.0	257.0
n300B	22473	22487	22529.4	64.0	n500G	26171	26196	26470.3	238.7
n300C	21183	21522	21618.2	81.6	n500H	35805	35750	35805.7	56.5
n300D	25220	25344	25450.6	128.6	n500I	30247	30097	30163.3	63.7
n300E	26636	26351	26744.0	173.3	n500J	30428	29925	30218.3	261.5

Table 3. Comparison of average results ($Q=10$)

n	GA		VNS/SA		OCPD	
	Best	Avr	Best	Avr	Best	Avr
100	12841.2	12954.1	12861.8	13091.1	12781.5	12825.8
200	19080.3	19339.4	18926.8	19151.4	18843.8	18954.2
300	23886.8	24224.2	23410.6	23683.1	23379.2	23618.7
400	26471.2	26748.7	26059.3	N/A	26016.5	26239.6
500	30245.6	30894.8	29579.4	N/A	29298.6	29524.0
Average	22505.0	22832.2	22167.5	N/A	22063.9	22232.4

Table 4. Comparison of the average running times ($Q=10$)

n	GA	VNS/SA	OCPD
100	21.1	542.2	166.3
200	95.2	N/A	1613.8
300	212.5	N/A	5009.6
400	358.2	N/A	11788.2
500	570.1	151103.0	21818.6

At the small instances (n in $\{20, 30, 40, 50, 60\}$ and $Q=10$) OCPD achieved the known optimum in 22 cases by the 50 cases. The difference of the average solutions from the optimum was less than 2% at 44 cases. At the other cases the difference was between 2 and 3.5%. The average running time was 35.7 seconds. These results show that we can use OCPD for the small instances too. Probably with other parameter selection we can improve the quality of the results.

6 Summary

In this paper we present a new evolutionary algorithm for the one-commodity pickup and delivery travelling salesman problem. This EA is a steady-state memetic algorithm that uses truncation selection, uses only mutation for the generation of the descendants and improves the solutions with three local search procedures. The mutation is based on “the EVL method”. As new techniques in the algorithm we use the EVL technique to memorize how strong the capacity condition was violated, we developed a new local search procedure and we apply a sequence of different local search procedures in the algorithm.

The algorithm was tested on large size benchmark problems; the quality of the results is good, the algorithm outperforms the previous heuristics in 84% of the large size test cases. We reached these good results with combined application of different techniques:

- the use of the three local search procedures one after the other lead to a more accurate search in the neighborhood of the descendent and we reached the solutions with smaller standard deviations,
- the use of the EVL technique with tree matrices (short term and long term memories of relative frequencies of the different values of the variables; and the memory of the

measure of the violation of the constraint) also improved the probability of reaching the best solution, and

- the reinsertion operation with the crowding technique helped to avoid local optima, it ensured good convergence.

The running time of the algorithm is large, but it is able to find better solutions than the earlier heuristics. In our future work we can probably improve these large running times with some shortcuts in the local search procedures and with reducing the repeating cycles in the local search procedures based on problem specific knowledge.

Acknowledgements The Hungarian Research Foundation OTKA K 68137 supported the study.

References

1. Anily S., Bramel J., Approximation algorithms for the capacitated traveling salesman problem with pickup and deliveries. *Naval Research Logistics*, 46, 654–670. (1999).
2. Borgulya I: Constrained optimization using a clustering algorithm. *Central European Journal of Operations Research*. 2000. 8 (1) 13-34.
3. Borgulya I., An Algorithm for the Capacitated Vehicle Routing Problem with Route Balancing. *Central European Journal of Operations Research*. Vol. 16 No. 4. 331-344 (2008).
4. Borgulya I., An Island Model for the No-Wait Flow Shop Scheduling problem. In: Schaefer et al. (Eds.): PPSN XI, Part II, LNCS 6239, 280-289. Springer Berlin, (2010).
5. Hernández-Pérez H., Salazar-González JJ, The one-commodity pickup-and-delivery travelling salesman problem. In: Jünger M, Reinelt G, Rinaldi G (eds.) *Combinatorial Optimization – Eureka, You Shrink!* LNCS 2570, 89–104. Springer Berlin, (2003).
6. Hernández-Pérez H., Salazar González JJ., A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145, 126–139. (2004a).
7. Hernández-Pérez H., Salazar González JJ., Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, 38, 245–255. (2004b).
8. Hernández-Pérez H., Rodríguez-Martín I., Salazar González JJ., A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research*, 36. 1639–1645. (2008).
9. Hosny MI, Mumford CL., Solving the One-Commodity Pickup and Delivery Problem Using an Adaptive Hybrid VNS/SA Approach. In: Schaefer et al. (Eds.): PPSN XI, Part II, LNCS 6239, 189-198, Springer Berlin, (2010).
10. Parragh SN., Doerner KF., Hartl RF., A survey on pickup and delivery problems Part II: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58 (2), 81–117. (2008)
11. Puchinger J., Raidl GR., Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation*, Springer LNCS 3562, pp. 41-53. (2005).
12. Raidl GR., A Unified View on Hybrid Metaheuristics, In: *Proc. of the 3rd Workshop on Hybrid Metaheuristics*, Gran Canaria, Spain, Springer LNCS 4030, pp.1-12, (2006).
13. Zhao F., Li S., Sun J., Mei D., Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. *Computer & Industrial Engineering* 56(4), 1642-1648, (2008).

An MILP-approach to the optimization of event-bus schedules: a scheduling application in the tourism sector

Gianluca Brandinu · Norbert Trautmann

Abstract This paper deals with “The Enchanted Journey”, which is a daily event tour booked by Indian tourists. During the tour, the participants visit original sites of famous Bollywood films at various locations in Switzerland; moreover, the tour includes stops for lunch and for shopping. Each day, up to five busses operate the tour. For operational reasons, however, two or more busses cannot stay at the same location simultaneously. Further operative constraints include time windows for activities and precedence constraints between activities. The planning problem consists in computing a feasible schedule for each bus. We implement a two step hierarchical approach. In the first step, we minimize the total waiting time, and in the second step, we minimize the total travel time of all busses. We present a formulation of this problem as a mixed-integer linear program, and we report on computational results obtained from the Gurobi solver.

1 Introduction

The tourism sector plays a major role for the economy in many countries, such as Switzerland, where in 2005 this sector added a gross value of 12.6 billion Swiss francs (cf. Baumann et al. [1]). Offers and demand in this sector are strongly influenced by the ongoing change from a leisure society to an experience society. Today, there is an increasing demand for holiday packages with event-type elements (cf. Müller [8]). An example for such elements are event tours, which let the participants transform a passive visit into experienced reality.

Here, we consider the event tour “The Enchanted Journey”. This tour leads the participants through famous sites of Bollywood films located in the mountainous region of Switzerland. The transportation of the participants between the sites is done by bus. Depending on the demand, up to five busses operate on the same day; however, there must not be several busses at the same location simultaneously. The problem discussed in this paper consists in computing a schedule for each bus operating on a given day such that the total waiting time (primary objective) and the total travel time (secondary objective) are minimized.

This problem is related to the multiple traveling salesman problem (cf. e.g. Bektas [2] and also Solomon [9]) and to the open-shop scheduling problem (cf. e.g. Brucker et al. [3]).

Gianluca Brandinu and Norbert Trautmann
Department of Business Administration, University of Bern, Schützenmattstrasse 14, 3012 Bern, Switzerland
E-mail: gianluca.brandinu@pqm.unibe.ch

Activity k	Location	Abbreviation	Type	Duration [min]
1	Engelberg	ENG	Start	0
2	Bern	BRN	Photo	30
3	Bern	BRN	Shopping	30
4	Zweisimmen	ZWS	Break	20
5	Montbovon	MBV	Photo	30
6	Rossinière	ROS	Photo	30
7	Saanen	SAA	Photo	30
8	Saanen	SAA	Photo	30
9	Gstaad	GST	Photo	15
10	Rougemont	ROU	Photo	30
11/12	Rougemont	ROU	Lunch L1/L2	30
13	Engelberg	ENG	End	0

Table 1 Activities in the tour

Dorndorf et al. [5], Graves [6], and Williamson et al. [11]). To the best of our knowledge, no extension of those problems has been discussed in the literature that captures all the specific constraints of the problem considered in this paper. Mellentien et al. [7] have presented a specific branch-and-bound algorithm for a similar problem arising in the context of event-marketing.

In this paper, we formulate the scheduling problem at hand as a mixed-integer linear program. This approach offers some flexibility with respect to changes of the problem specification, i.e., the design of the tour, which was required by the tour operator. Our computational results show that all relevant problem instances can be (approximately) solved in short CPU time.

The remainder of the paper is structured as follows. In Section 2, we describe the event tour in more detail, and we state the arising scheduling problem. In Section 3, we present the MILP formulation. In Section 4, we report on our numerical results. Concluding remarks and suggestions for further research follow in Section 5.

2 Real-World Application and Scheduling Problem

In Subsection 2.1, we present the real-world application that we deal with in this paper. In Subsection 2.2, we state the scheduling problem arising from this application.

2.1 Real-World Application

We refer to a bus tour which is offered daily in Switzerland and booked by tourists from India. During the tour, the participants visit seven original sites of various Bollywood films. These sites are located in the city of Bern and in the surrounding mountains. At each site, the participants are photographed while imitating famous scenes from these films. Moreover, each tour comprises a lunch break and a guided shopping tour in Bern. In total, the tour consists of ten activities, whose prescribed durations vary between 15 and 30 minutes (cf. Table 1).

The town Engelberg in central Switzerland is the starting point as well as the ending point of the tour. The activities sequence of the tour is not pre-determined. However, Bern must always be either the first or the last location. Moreover, for the entire tour and for

t [min]	ENG	BRN	ZWS	MBV	ROS	SAA	GST
ENG	–						
BRN	125	–					
ZWS	161	81	–				
MBV	196	71	46	–			
ROS	196	82	35	11	–		
SAA	178	98	17	29	18	–	
GST	183	103	22	34	23	05	–
ROU	184	94	23	23	12	06	11

Table 2 Travel times between the locations

each activity, respectively, a time window is prescribed, which arises e.g. from the service hours for lunch or the shop-opening hours. In addition, the two respective activities at Bern, Saanen, and Rougemont must be performed in a row.

Each day, up to five busses operate this tour. There are two alternative locations (L1 and L2) for lunch, and every bus must stop at exactly one of them. All other locations must be part of every tour, and at each of them there may be only one bus at a time; otherwise, the participants could not be photographed individually. For each location, five minutes buffer time between succeeding busses is prescribed.

Due to the geographical location of the sites the transfer between the locations is carried out by busses. The corresponding travel times are given in Table 2. Due to law and regulations in Switzerland, the total travel time of any bus may not exceed 540 minutes. A shortest route of a bus corresponds to a total travel time of 413 minutes. The total duration of the tour may not exceed 810 minutes.

The tour operator wishes to perform the tours of each day such that primarily the total waiting time of all busses is minimized. As secondary objective, the tour operator wishes to minimize the total travel time of the busses.

2.2 Scheduling Problem

For each bus, the tour operator decides manually the location where lunch is served, and whether Bern is the first or the last location.

Given the number of busses to be planned, the scheduling problem discussed in this paper is then as follows. Sought is an individual schedule for each bus such that all organizational and logistic constraints mentioned in Subsection 2.1 are fulfilled, and the total waiting time of all busses (primary objective) and the total travel time (secondary objective) are minimized.

3 The MILP Formulation

In this section, we formulate the scheduling problem introduced in Subsection 2.2 as a mixed-integer linear program. In our formulation, we interpret the activities of the tour as the processors and the busses as the jobs of an open-shop scheduling problem.

In Subsection 3.1, we introduce the decision variables used in our model. In Subsection 3.2, we formulate the objective functions. In Subsection 3.3, we describe the constraints.

3.1 Variables

Similar to the machine scheduling models presented by Chen [4] and Wagner [10], we use the following decision variables. Let $s_{ijk} \geq 0$ denote the start time of activity k at position j for bus i . The activities sequence of bus i is controlled by the binary variable x_{ijk} with $x_{ijk} = 1$ if activity k is scheduled at position j for bus i , and $x_{ijk} = 0$ otherwise. The bus sequence of activity k is modeled by the binary variable $z_{ii'k}$ with $z_{ii'k} = 1$ if bus i starts activity k later than bus $i' \neq i$, and $z_{ii'k} = 0$ otherwise. Moreover, let $W_{ij} \geq 0$ and $T_{ij} \geq 0$ denote the waiting time and the travel time, respectively, of bus i before the activity at position j .

3.2 Objective Function

For modeling reasons, we introduce two fictitious activities $k = 1$ and $k = 13$ representing the start and the end, respectively, of the tour. Recall that every bus stops at only one of the two places where lunch is served; we represent this by two respective activities. Thus, in total there are $m = 13$ activities, and the tour of each bus tour consists of $N = 12$ activities. Let $r_{ik} = 1$ if activity k is part of the tour of bus i , and $r_{ik} = 0$ otherwise. By n we denote the number of busses. Furthermore, let d_{ik} be the duration of activity k for bus i , and $t_{kk'}$ be the travel time from activity k to activity k' .

For the waiting time W_{ij} of bus i before the activity at position j , constraint (1) must hold.

$$W_{ij} \geq s_{ijk'} - (s_{i,j-1,k} + d_k + t_{kk'}) - M(2 - x_{ijk'} - x_{i,j-1,k})$$

$$(i = 1, \dots, n; j = 2, \dots, N; k, k' = 1, \dots, m; k \neq k')$$
 (1)

For the travel time T_{ij} , constraint (2) must be fulfilled.

$$T_{ij} \geq (x_{i,j-1,k} + x_{ijk'} - 1)t_{kk'}$$

$$(i, i' = 1, \dots, n; j = 2, \dots, N; k, k' = 1, \dots, m; k \neq k')$$
 (2)

In order to account for the two-level planning objective, we apply a two-step hierarchical approach as follows. First we minimize the total waiting time:

$$\text{Min.} \quad \sum_{i=1}^n \sum_{j=2}^N W_{ij}$$

Let W^* denote the resulting total waiting time. Then, we minimize the total travel time

$$\text{Min.} \quad \sum_{i=1}^n \sum_{j=2}^N T_{ij}$$

subject to

$$\sum_{i=1}^n \sum_{j=2}^N W_{ij} \leq W^*$$

3.3 Constraints

We link the variables s_{ijk} and x_{ijk} by

$$s_{ijk} \leq Mx_{ijk} \quad (i = 1, \dots, n; j = 1, \dots, N; k = 1, \dots, m) \quad (3)$$

The tour of each bus consists of the selected activities, i.e.,

$$x_{ijk} \leq r_{ik} \quad (i = 1, \dots, n; j = 1, \dots, N; k = 1, \dots, m) \quad (4)$$

$$\sum_{j=1}^N x_{ijk} = r_{ik} \quad (i = 1, \dots, n; k = 1, \dots, m) \quad (5)$$

$$\sum_{k=1}^m x_{ijk} = 1 \quad (i = 1, \dots, n; j = 1, \dots, N) \quad (6)$$

The time lag between the start of activity k and the start of the subsequent activity k' must not be smaller than the duration d_{ik} of activity k plus the travel time $t_{kk'}$, i.e.,

$$s_{ijk} + d_{ik} + t_{kk'} \leq s_{i,j+1,k'} + M(2 - x_{ijk} - x_{i,j+1,k'}) \quad (i = 1, \dots, n; j = 1, \dots, N-1; k, k' = 1, \dots, m; k \neq k') \quad (7)$$

For each activity k , the sequence of the busses must be determined:

$$z_{ii'k} + z_{i'ik} = 1 \quad (i, i' = 1, \dots, n; k = 1, \dots, m; i \neq i') \quad (8)$$

The time lag between the start of activity k in the tour of bus i' and the tour of the subsequent bus i must not be smaller than the duration of k plus the buffer time b , i.e.,

$$s_{ijk} - s_{i'j'k} \geq d_{ik} + b - M(2 - x_{ijk} - x_{i'j'k}) - M(1 - z_{ii'k}) \quad (i, i' = 1, \dots, n; j = 1, \dots, N; j' = 1, \dots, N; k = 1, \dots, m; i \neq i') \quad (9)$$

We formulate the further organizational and logistic constraints mentioned in Subsection 2.2 as follows. Each bus tour starts and ends in Engelberg, i.e.,

$$x_{i11} = 1 \quad (i = 1, \dots, n) \quad (10)$$

$$x_{iN,13} = 1 \quad (i = 1, \dots, n) \quad (11)$$

The two activities $k = 2$ and $k = 3$ in Bern are the first ($\xi_i = 1$) or the last ($\xi_i = 0$) activities in each tour, i.e.,

$$2\xi_i = \sum_{k=2}^3 (x_{i2k} + x_{i3k}) \quad (i = 1, \dots, n) \quad (12)$$

$$2 - 2\xi_i = \sum_{k=2}^3 (x_{i,10,k} + x_{i,11,k}) \quad (i = 1, \dots, n) \quad (13)$$

Each activity k must be started within its time window $[ES_k, LS_k]$, i.e.,

$$ES_k \leq s_{ijk} + M(1 - x_{ijk}) \quad (i = 1, \dots, n; j = 1, \dots, N; k = 1, \dots, m) \quad (14)$$

$$LS_k \geq s_{ijk} \quad (i = 1, \dots, n; j = 1, \dots, N; k = 1, \dots, m) \quad (15)$$

Both activities in Saanen as well as both activities in Rougemont have to be visited directly after each other. We therefore introduce the constraints

$$x_{ij7} \leq x_{i,j-1,8} + x_{i,j+1,8} \quad (i = 1, \dots, n; j = 2, \dots, N-1) \quad (16)$$

$$x_{ij8} \leq x_{i,j-1,7} + x_{i,j+1,7} \quad (i = 1, \dots, n; j = 2, \dots, N-1) \quad (17)$$

$$x_{ij10} \leq x_{i,j-1,11} + x_{i,j+1,11} + x_{i,j-1,12} + x_{i,j+1,12} \\ (i = 1, \dots, n; j = 2, \dots, N-1) \quad (18)$$

$$x_{ij11} + x_{ij12} \leq x_{i,j-1,10} + x_{i,j+1,10} \\ (i = 1, \dots, n; j = 2, \dots, N-1) \quad (19)$$

The total travel time of any bus i may not exceed 540 minutes, i.e.,

$$\sum_{j=2}^N T_{ij} \leq 540 \quad (i = 1, \dots, n) \quad (20)$$

and the total duration of any tour may not exceed 810 minutes, i.e.,

$$\sum_{j=1}^N s_{ij,13} - \sum_{j'=1}^N s_{ij',1} \leq 810 \quad (i = 1, \dots, n) \quad (21)$$

The total travel time of bus i cannot be smaller than the total travel time in a shortest route, i.e. smaller than 413 minutes:

$$\sum_{j=2}^N T_{ij} \geq 413 \quad (i = 1, \dots, n) \quad (22)$$

4 Computational Results

In this section, we report on the results of our experimental performance analysis. In Subsection 4.1, we describe the problem instances used for the experiment. In Subsection 4.2, we present our computational results.

4.1 Problem Instances

Based on the data and some scenarios provided by the tour operator, we have generated 32 problem instances by varying the number of busses n between 2 and 5 and the respective values for the parameters ξ_i and r_{ik} , which indicate whether the activities in Bern are the first or the last of the tour, and which location is used for lunch. Table 3 summarizes this data for the 32 instances. The columns of this table indicate the number of the problem instance, the number of busses, and the values for parameter ξ_i and the lunch location for each bus.

Instance	n	ξ_i ($i = 1, \dots, n$)	Lunch location ($i = 1, \dots, n$)	Instance	n	ξ_i ($i = 1, \dots, n$)	Lunch location ($i = 1, \dots, n$)
1	2	1,1	L2,L1	17	4	0,1,1,1	L2,L1,L1,L2
2	2	1,0	L2,L1	18	4	1,1,0,0	L2,L1,L1,L2
3	2	0,1	L2,L1	19	4	1,0,1,0	L2,L1,L1,L2
4	2	0,0	L2,L1	20	4	1,0,0,1	L2,L1,L1,L2
5	3	1,1,1	L2,L1,L1	21	4	0,1,1,0	L2,L1,L1,L2
6	3	1,1,0	L2,L1,L1	22	4	0,1,0,1	L2,L1,L1,L2
7	3	1,0,1	L2,L1,L1	23	4	0,0,1,1	L2,L1,L1,L2
8	3	0,1,1	L2,L1,L1	24	4	1,0,0,0	L2,L1,L1,L2
9	3	1,0,0	L2,L1,L1	25	4	0,1,0,0	L2,L1,L1,L2
10	3	0,1,0	L2,L1,L1	26	4	0,0,1,0	L2,L1,L1,L2
11	3	0,0,1	L2,L1,L1	27	4	0,0,0,1	L2,L1,L1,L2
12	3	0,0,0	L2,L1,L1	28	4	0,0,0,0	L2,L1,L1,L2
13	4	1,1,1,1	L2,L1,L1,L2	29	5	1,1,0,0,1	L2,L1,L1,L2,L1
14	4	1,1,1,0	L2,L1,L1,L2	30	5	0,0,1,0,1	L2,L1,L1,L2,L1
15	4	1,1,0,1	L2,L1,L1,L2	31	5	0,0,0,1,1	L2,L1,L1,L2,L1
16	4	1,0,1,1	L2,L1,L1,L2	32	5	0,0,0,1,0	L2,L1,L1,L2,L1

Table 3 Problem instances

4.2 Results

We have implemented the model presented in Section 3 in AMPL. For the computational experiment, we have used version 4.0 of the Gurobi Solver on a standard PC (Intel Core Quad CPU with 3GHz clock speed and 8GB RAM). For each problem instance, we have prescribed a CPU time limit of 10'800 seconds for the minimization of the total waiting time, and a CPU time limit of 10'800 seconds for the subsequent minimization of the total travel time (cf. Subsection 3.2), respectively.

Table 4 summarizes the results we obtained. For each instance, the table indicates the relative integrality gap and the CPU time used for the minimization of the total waiting time and the total travel time, respectively. The entry "limit" indicates that the solver has stopped due to the prescribed CPU time limit.

For each instance we obtained a feasible solution within the prescribed time limit. Moreover, all instances could be solved to optimality with respect to the total waiting time. One reason for that might be that for each instance, there exists a solution with zero total waiting time. The subsequent minimization of the total travel time poses a harder problem; here, we obtained proven optimal solutions for 23 of the 32 problem instances. Figure 1 shows an optimal solution to instance 32. In this solution, the total waiting time is zero, and the total travel time of each bus coincides with the total travel time on a shortest route.

In a further analysis, we have moderately increased or decreased the travel times or the activities durations in selected problem instances. With respect to the total waiting time, all modified instances could be solved to optimality within reasonable CPU times. With respect to the total travel time, the results are very similar to those shown in table 4.

5 Summary and Outlook

In this paper we have been concerned with "The Enchanted Journey", a real-world scheduling application in the event-tourism sector. We have formulated the scheduling problem as a mixed-integer linear program. For a computational analysis, we have generated 32 problem instances by varying the number of busses and some problem-specific parameters. The

Instance	Waiting time		Travel time		Instance	Waiting time		Travel time	
	Gap [%]	CPU time [sec]	Gap [%]	CPU time [sec]		Gap [%]	CPU time [sec]	Gap [%]	CPU time [sec]
1	0	4	0	6	17	0	262	1.5	limit
2	0	4	0	5	18	0	117	0	986
3	0	1	0	5	19	0	107	0	1'148
4	0	2	0	4	20	0	123	0	215
5	0	37	0.4	limit	21	0	105	0	1'139
6	0	9	0	43	22	0	123	0	693
7	0	5	0	72	23	0	101	0	307
8	0	12	0	12	24	0	92	0	518
9	0	8	0	84	25	0	138	0	171
10	0	11	0	34	26	0	22	0	374
11	0	12	0	62	27	0	141	0	434
12	0	10	0	50	28	0	348	0	437
13	0	1'047	13.3	limit	29	0	3'816	9.9	limit
14	0	241	2.4	limit	30	0	1'243	11.4	limit
15	0	130	2.4	limit	31	0	269	1.2	limit
16	0	343	0.7	limit	32	0	7'761	0	9'748

Table 4 Results

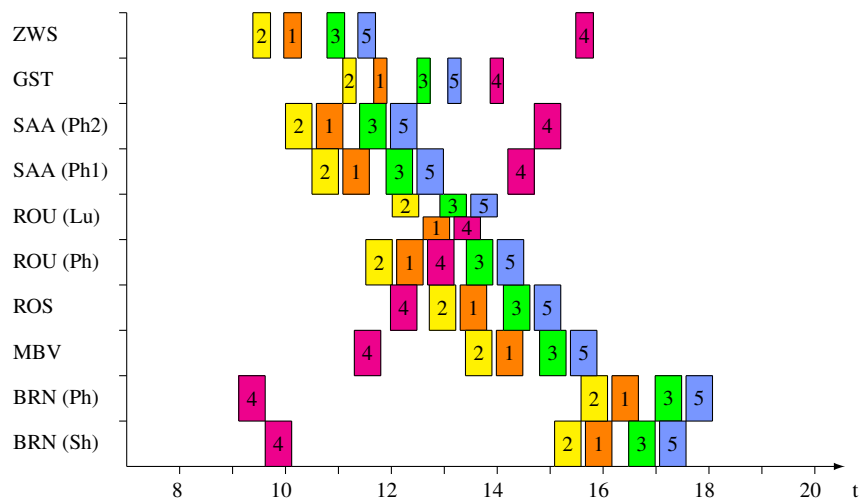


Fig. 1 Optimal schedule for 5 busses

results, which we obtained using AMPL and the Gurobi Solver 4.0, demonstrate the applicability and the robustness of our approach.

Currently, the tour operator wishes to decide upon the lunch location and the position of Bern in each Tour manually (cf. Subsection 2.2). It should be investigated how the model presented in Section 3 can be extended such that these decisions are included efficiently in the optimization.

Further improvements of the CPU time requirements may be achieved by introducing symmetry-breaking constraints, i.e., constraints that eliminate schedules from the search space that are identical to other schedules except the numbering of the busses. Moreover, it should be studied whether it would be advantageous to combine the two objectives (to-

tal waiting time and total travel time) into a single objective function, e.g. by putting a sufficiently large weight on the total waiting time. Eventually, the proposed MILP approach should be compared against other exact approaches (e.g., constraint programming) or heuristics; for the latter, a possible direction for future research is to extend existing heuristics for related problems, in particular for the multiple traveling salesman problem and the open-shop scheduling problem, by the specific constraints of the problem studied in this paper.

References

1. Baumann, T., Schiess, U.: Satellitenkonto Tourismus der Schweiz, 2001 und 2005 – Grundlagen, Methodik und Ergebnisse. Bundesamt fuer Statistik, Neuenburg (2008)
2. Bektas, T.: The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega* **34**, 209 – 219 (2006)
3. Brucker, P., Knust, S., Cheng, T., Shakhlevich, N.: Complexity results for flow-shop and open-shop scheduling problems with transportation delays. *Annals of Operations Research* **129**, 81–106 (2004)
4. Chen, J.S.: An Integer Programming Model for the Open Shop Scheduling Problem (2003). Working Paper
5. Dorndorf, U., Pesch, E., Phan-Huy, T.: Solving the open shop scheduling problem. *Journal of Scheduling* **4**, 157–174 (2001)
6. Graves, S.: A review of production scheduling. *Operations Research* **29**, 646–675 (1981)
7. Mellentien, C., Schwindt, C., Trautmann, N.: Scheduling the factory pick-up of new cars. *OR Spectrum* **26**, 579–601 (2004)
8. Mueller, H.: Tourism and Hospitality into the 21st Century. In: A. Lockwood, S. Medlik (eds.) *Tourism and Hospitality in the 21st Century*, pp. 61–70. Butterworth Heinmann, Oxford (2001)
9. Solomon, M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* **35**, 254–265 (1987)
10. Wagner, H.: An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly* **6**, 131–140 (1959)
11. Williamson, D.P., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevast'janov, S.V., Shmoys, D.B.: Short shop schedules. *Operations Research* **45**, 288–294 (1997)

A more realistic approach for airport ground movement optimisation with stand holding

Jason A. D. Atkin · Edmund K. Burke ·
Stefan Ravizza

Abstract Airports are already facing several challenges and can expect to face more in the future. Despite the requirements to handle ever increasing numbers of aircraft, they also have to meet environmental targets and regulations. Although physical expansion of the airport could sometimes help, this is rarely possible in practice. The complexity increases the closer an airport has to work to their maximal possible capacity. The complexity of the problem means that advanced decision support systems are needed to guarantee efficient airside airport operations and to mitigate the environmental impact. This research considers the important problem of getting aircraft from source to destination locations (usually either runways or gates/stands) in as efficient a manner as possible, in terms of time or fuel burn. A new sequential graph-based algorithm is introduced for this important part of the airside operations at an airport - usually named the ground movement problem. This algorithm, embedded in a wider operational system, has several advantages over previous approaches for increasing the realism of the modelling and it utilises a recently developed approach to more accurately estimate taxi speeds. Importantly, the taxi time prediction and the ground movement model use the same directed graph representation of the airport. For the experiments in this paper, the algorithm has been configured to absorb as much of the waiting time as possible for departures at the gate/stand, to reduce the fuel burn and the environmental impact. Analysis with data from a European hub airport shows very promising results and gives an indication of both the performance of the system (in comparison to a lower bound on the taxi time) and the limits to the amount of waiting time which could possibly be absorbed as stand hold (without the engines running).

1 Introduction

European airports face several challenges in the 21st century, including the capacity challenge (with demands for air travel still increasing year on year) and the environmental challenge [1]. To avoid forming huge bottlenecks in the air transportation

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

system, airports have either to be enlarged, or (since enlargement is not possible in most cases), to utilise the existing resources as efficiently as possible. In addition, the increasing focus upon environmental issues is likely to further grow over time.

As airports work closer to their maximum capacity, airside airport operations become much harder to deal with. As a result, decision support systems have to be increasingly advanced and need to both integrate different airside airport operations with each other and also model each process increasingly realistically.

From an optimisation point of view, ground movement of aircraft can be considered to be one of the most important airside operations at an airport [6], since it links several other problems together, such as the runway sequencing problems for arrivals and/or departures [4], the stand holding problem [2,3] and the gate assignment problem [11]. For a comprehensive literature review of ground movement research and the integration with other operations, we point the interested reader to a recently published review [6].

This paper presents a decision support framework for environmentally friendly ground movement and promising experimental results which utilise more realistic taxi time predictions for a European hub airport. A framework is described for integrating a graph-based sequential movement algorithm into a larger decision support system which can also consider the runway sequencing problem and the stand holding problem. A statistical approach has been used to more accurately estimate taxi times for aircraft than a standard lookup table may allow. This utilises the same graph which is used for the ground movement model. This integrated approach allows the effects of ground plan changes to be modelled more accurately, changing both taxi speed predictions and routing information. In addition, several concepts have been included in the model which allow airport layouts to be modelled in a more realistic manner, such as restricting certain taxiways to be used only by certain aircraft and coping with the required separations between aircraft. Finally, the absorption of delay at the stand, prior to starting the engines, has been considered. This reduces the waiting times at the runway and is further extending previous stand holding ideas [2,3,8]. The maximal potential benefits of such a system have been quantified.

Section 2 provides a description of the airport ground movement problem and how it can be embedded into the larger combined sequencing/routing/stand holding framework. The test dataset from the airport is then presented in Section 3 together with the statistical estimation of taxi times. Following this, the sequential ground movement algorithm which has been developed, and was utilised for these experiments, is detailed in Section 4. The results of the application of the algorithm to the test dataset are then shown in Section 5; before the paper ends with some conclusions in Section 6.

2 Problem description

The links between the ground movement problem and runway sequencing are considered first in this section, before the ground movement problem itself is discussed in more detail. The section ends with a consideration of the stand holding benefits which can result from the appropriate solution of the ground movement problem.

2.1 The links with runway sequencing

Atkin et al. [6] highlighted the importance of integrating the ground movement problem with other airside airport operations, such as the problems of finding good departure and arrival sequences. Supporting controllers in these tasks is a challenge, especially when departures and arrivals have common restrictions and interactions due to the airport layout. For this paper, we assume that the runway sequencing and ground movement problems are solved as two distinct stages. The integrated (departures and arrivals) runway sequencing problem is solved in a first stage, then the consequent landing and take-off times are used in the second stage within the consideration of the ground movement problem. Thus, the wheels-on time at the runway (for arrivals) and the wheel-off time at the runway (for departures) are both assumed to be fixed within the ground movement problem. Later research will analyse the benefits of providing a feedback loop from the ground movement problem to the integrated runway sequencing problem and of closer integration between the two problems.

2.2 Problem description of the ground movement problem

This paper considers ground movement at an airport. The ground movement problem is a combined routing and scheduling problem. It involves guiding aircraft on the surface of an airport to their destinations in a timely manner, where the goal is to reduce the overall travel time and to enable the target take-off times at the runway to be met. It is important, for reasons of safety that two aircraft never conflict with each other throughout the ground movement process.

In the model which is considered in this paper, the route of the aircraft is not pre-determined, allowing greater flexibility for solutions; however, the utilised solution method provides the possibility to restrict certain aircraft to specific taxiways and/or to avoid routes which involve tight turns. The airport layout is represented as a directed graph, where the edges represent the taxiways and the vertices represent the junctions or intermediate points. Aircraft are considered to occupy edges, and conflicts are avoided by preventing any two aircraft from using the same edge simultaneously.

As we will discuss in the next section, the aircraft movement speed depends upon various factors and, in order to provide a more realistic ground movement decision support system, it is important to take this variation into account. The times at the runway are assumed to be fixed for departures as well as for arrivals. The sequential approach to ground movement will then minimise the taxi time for each individual aircraft given the movement of the aircraft which have already been routed. Hence, the approach will attempt to absorb as much of the waiting time as possible at the gate/stand. Consequently, the aircraft can start its engines as late as possible, reducing fuel burn and environmental impact. Thus, the solution method can be considered to be not only reducing the ground movement time, but also solving the stand holding problem [2, 3, 8].

3 Analysed case: Zurich Airport

This analysis utilised data from Zurich Airport (ZRH), which is the largest airport in Switzerland and a hub airport for Swiss International Air Lines. The airport has three

runways, named 10/28, 14/32 and 16/34, according to their direction of operation, with two runways intersecting each other (see Figure 1). The considered data included information about the airport layout, the positions of stand and runway entrance and exit points and the layouts of all of the taxiways as well as the real timings for the various aircraft. This information was used to develop a taxi time prediction function, as discussed below, to improve the accuracy of the taxi time predictions which are used in ground movement. We had access to data for an entire day's operations for the 19th of October 2007. No extraordinary occurrences took place and there were 679 movements in total (337 arrivals and 342 departures).

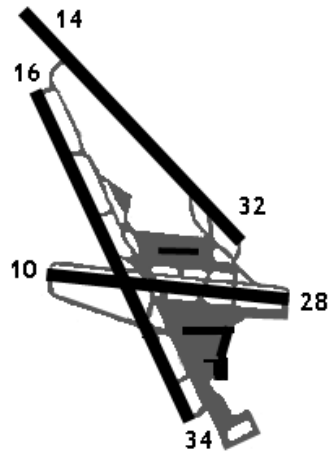


Fig. 1 Sketch of Zurich Airport (ZRH)

Ground movement models need taxi time prediction, but sufficiently accurate values are rarely available. In previous research, average taxi speeds have been assumed to be influenced only by the aircraft type. However, other research has shown that the aircraft type is not the most influencing factor upon taxi speeds [5,13,17]. Comparisons between ground movement tool results and the status quo at airports have previously been hard to analyse, due to the need for accurate taxi speed data, since historic data usually includes the effects of any delays or re-routing due to routing/timing conflicts between aircraft. Thus, the effects of taxi time variability and the benefits from the ground movement decision support system were often intermingled.

An approach to more accurately predict taxi times for aircraft or, equivalently, their average speeds, was proposed in Atkin et al. [5]. Multiple regression was used to estimate a function which could more accurately predict the taxi speed and identify the factors which were most related to taxi speeds, such as total distance travelled, total turning angle and the number of other aircraft of different types which were on the airport's surface. The aim was to then eliminate the effects of factors which represented the actual amount of traffic at the airport, with the goal being to predict the taxi times for unimpeded aircraft. These could be used in a more advanced ground movement decision support system, such as the one described in this paper, providing the opportunity to compare results with the way in which an airport was actually operating. Such a system will itself model the effects of the interaction between aircraft (so

these should not be included in its taxi speed data). The resulting taxi time prediction functions were utilised to predict the taxi times which were employed in this paper.

Depending upon the terminal and the operating mode (which runways are in use), runway crossings may be necessary during the taxi process. We plan to integrate these effects into the combined ground movement and sequencing model later, but for the moment they are included in the statistical model for taxi speeds.

4 Ground movement decision support system

In the routing approach described in this paper, the aircraft are routed sequentially. When an aircraft is ready, it has to be routed respecting previous reservations by other aircraft using the taxiways. The routes which have been previously calculated for other aircraft do not change as new aircraft are taken into consideration. Thus to acknowledge the difficulty and time costs associated with communicating changes to pilots and reducing the quantity of communication needed between the surface controllers and the pilots. The objective for each of the sequential routings is to find the routing with minimal taxi time among all remaining conflict-free routings.

The approach described here is based on research by Gawrilow et al. [12] and the PhD thesis of Stenzel [21]. Ravizza modified the approach for his Master's dissertation [18] to label the vertices instead of the edges to simplify their interpretations. The original aim of this approach was to control automated guided vehicles in container terminals in harbours or in storage areas but is here applied instead to routing aircraft. The approach has then been further modified for this work, for instance by allowing the approach to work backwards to meet a specified end time rather than starting time. The resulting algorithm is described in this section.

A directed graph model of the airport is used by the algorithm. The so called Quickest Path Problem with Time Windows (QPPTW) algorithm is a generalized vertex-based label-setting algorithm based on Dijkstra's algorithm and can sequentially route aircraft on the airport surface. In contrast to many other ground movement support systems which use time discretisation [7, 15, 16, 19], no such discretisation is used in this approach. It has similarities to the recently published work by Lesire [14], which used a sequential A* algorithm, but it provides a better coverage of the solution space, potentially allowing it to find better solutions within comparable execution times - these being short enough for it to be appropriate for real-time decision making.

The algorithm which is used in this paper also provides the possibility to define which edges in the graph are in conflict with each other and cannot be used simultaneously. In addition, for each edge incident to a vertex, the set of valid outgoing edges can be manually defined if desired, or can depend upon information about the aircraft. This enables the decision support system to forbid aircraft from making narrow turns or prevent aircraft from using taxiways for which they are too large. Together, these features enable the approach to more realistically model the airport surface while leaving the routing task itself to the algorithm.

The preprocessing of the algorithm is explained in Section 4.1, before introducing the QPPTW algorithm. The section ends with the discussion about buffer times and the sequencing of the routed aircraft.

4.1 Ground plan preprocessing

It is important to maintain separations between aircraft on the ground. The concept of conflicting edges is introduced here for this reason, so that no two conflicting edges can be occupied simultaneously. The conflicting edges are determined in a preprocessing stage. For this research, we used an approach which assumes straight connecting lines between vertices since this requires less time in the preprocessing stage and is adequate for the directed graph model which has been used in this research, where aircraft drive in an almost straight line between vertices. In this approach, two edges conflict with each other if they are located closer together than a given threshold distance. Edges in the graph, together with their embedding in the airport plan, are here named segments. To find the minimal Euclidian distance between two segments, the algorithm performs two processing steps. Firstly, it verifies whether the edges are intersecting, then, if they are disjoint, the distance between each end point of one segment and the closest point on the other segment is calculated. The minimum over these four distances corresponds to the minimal distance between the two segments.

4.2 Variable definitions

Definitions of the variables and data structures which are used in the model are given in Table 1.

Table 1 Table of definitions

Variable	Explanation
$confl(e)$	The set of edges which conflicts with edge $e \in E$
$F_e = [a_e, b_e]$	A time-window on edge $e \in E$
$\mathcal{F}(e)$	The sorted set of all the time-windows on edge $e \in E$
$G = (V, E)$	The directed graph representing the airport layout, with vertices $v \in V$ and edges $e \in E$
H	The Fibonacci heap storing the added labels
$I_L = [a_L, b_L]$	The time interval used in a label L
$L = (v_L, I_L, pred_L)$	A label on vertex $v_L \in V$ with time interval I_L and predecessor label $pred_L$
$\mathcal{L}(v)$	The set of all of the labels at vertex $v \in V$
R	A conflict-free route that is being generated
$T = (s, t, time)$	A taxi request to route, from source $s \in V$ at time $time$ to target $t \in V$
w_e	The weight (necessary taxi time) of edge $e \in E$

4.3 Key concepts

The QPPTW algorithm with its expansion steps works in a similar way to Dijkstra's algorithm [9,10]. However, in the QPPTW algorithm, a label can be expanded several times due to the different time-windows and a concept of dominance is needed in order

to guarantee a polynomial solution time. It is necessary to define some of the concepts upon which the approach is based. Firstly, the algorithm needs information about the times that each part of the taxiway (edge) is free and this is provided in the form of a sorted set of free time-windows for each edge:

Definition: Set of sorted time-windows

The set $\mathcal{F}(e)$ contains the sorted set of time intervals $F_e^j = [a_e^j, b_e^j]$ which specify the times when the edge e can be used for a new route. This will exclude the times when e , or an edge which conflicts with e , are in use by previously routed aircraft. These are inputs to the routing algorithm for each aircraft.

The use of labels is the essential concept of the QPPTW algorithm.

Definition: Label

A label $L = (v_L, I_L, pred_L)$ specifies the time period $I_L = [a_L, b_L]$ within which the current aircraft could reach vertex v_L . It includes a reference to the previous label on the route, $pred_L$, and thus implicitly represents a route (with edge traversal timings) from a source vertex to the specified vertex v_L . These labels are generated as the routing algorithm progresses, together specifying the (undominated) time periods (from time a_L to time b_L) when the current aircraft could reach vertex v_L .

An ordering relation is defined over the intervals of the labels to allow the definitions of dominance.

Definition: Dominance

A label L dominates a label L' on vertex $v_L = v_{L'}$ if and only if $I_{L'} \subseteq I_L$ (and there are identical route restrictions on the outgoing edges), which implies $a_L \leq a_{L'}$ and $b_L \geq b_{L'}$.

Once the routing has been performed by the QPPTW algorithm, the time-windows are readjusted (as discussed in Section 4.6) before the QPPTW algorithm is reapplied to route the next aircraft.

4.4 QPPTW algorithm

The input of the QPPTW algorithm contains the graph $G = (V, E)$ with its weight function w_e , which corresponds to the taxi times for each edge, estimated using the taxi time estimation method which was described in Section 3. The sorted set of available time-windows $\mathcal{F}(e)$ also has to be provided for each edge e , specifying when the edge is available. A taxi request $T_i = (s_i, t_i, time_i)$ for aircraft i is then a conflict-free route R from the vertices s_i to t_i with minimal taxi time (w.r.t. w_e) that respects the given time-windows.

The pseudocode of the QPPTW algorithm is shown in Algorithm 1 and is a close variant of the QPPTW algorithm described by Stenzel in [21]. The main difference is that we allocate the labels to vertices, which helps both to model the process more realistically and to more easily understand the algorithm, since it distinguishes between the use of the labels at the vertices and the input time-windows at the edges.

Lines 1 and 2 of Algorithm 1 involve the initialization of the Fibonacci heap and the references to the Fibonacci heap which are stored at each vertex. The use of Fibonacci

Algorithm 1: Quickest Path Problem with Time Windows (QPPTW)

Input: Graph $G = (V, E)$ with weights w_e for all $e \in E$, the set of sorted time-windows $\mathcal{F}(e)$ for all $e \in E$, a taxi request $T_i = (s_i, t_i, time_i)$ with the source vertex $s_i \in V$, the target vertex $t_i \in V$ and the start time $time_i$.

Output: Conflict-free route R from s_i to t_i with minimal taxi time that starts at the earliest at time $time_i$, respects the given time-windows $\mathcal{F}(e)$ or returns the message that no such route exists.

```

1  Let  $H = \emptyset$ 
2  Let  $\mathcal{L}(v) = \emptyset \quad \forall v \in V$ 
3  Create new label  $L$  such that  $L = (s_i, [time_i, \infty), nil)$ 
4  Insert  $L$  into heap  $H$  with key  $time_i$ 
5  Insert  $L$  into set  $\mathcal{L}(s_i)$ 
6  while  $H \neq \emptyset$  do
7    Let  $L = H.getMin()$ , where  $L = (v_L, I_L, pred_L)$  and  $I_L = [a_L, b_L]$ 
8    if  $v_L = t_i$  then
9      Reconstruct the route  $R$  from  $s_i$  to  $t_i$  by working backwards from  $L$ 
10     return the route  $R$ 
11   forall the outgoing edges  $e_L$  of  $v_L$  do
12     foreach  $F_{e_L}^j \in \mathcal{F}(e_L)$ , where  $F_{e_L}^j = [a_{e_L}^j, b_{e_L}^j]$ , in increasing order of  $a_{e_L}^j$  do
13       /* Expand labels for edges where time intervals overlap */
14       if  $a_{e_L}^j > b_L$  then
15         goto 11 /* consider the next outgoing edge */
16       if  $b_{e_L}^j < a_L$  then
17         goto 12 /* consider the next time-window */
18       Let  $time_{in} = \max(a_L, a_{e_L}^j)$  /*  $a_{e_L}^j > a_L \Rightarrow$  waiting */
19       Let  $time_{out} = time_{in} + w_{e_L}$ 
20       if  $time_{out} \leq b_{e_L}^j$  then
21         Let  $u = head(e_L)$ 
22         Let  $L' = (u, [time_{out}, b_{e_L}^j], L)$ 
23         /* dominance check */
24         foreach  $\hat{L} \in \mathcal{L}(u)$  do
25           if  $\hat{L}$  dominates  $L'$  then
26             goto 12 /* next time-window */
27           if  $L'$  dominates  $\hat{L}$  then
28             Remove  $\hat{L}$  from  $H$ 
29             Remove  $\hat{L}$  from  $\mathcal{L}(u)$ 
30         Insert  $L'$  into heap  $H$  with key  $a_{L'}$ 
31         Insert  $L'$  into set  $\mathcal{L}(u)$ 
32 Output: there is no  $s_i$ - $t_i$  route

```

heaps for this algorithm has the same beneficial effect upon the execution time as it does for Dijkstra's algorithm. The starting label is generated for the source s_i in line 3 and is then inserted into the Fibonacci heap, which is sorted with respect to the earliest possible arrival time (key). A reference is maintained to this label using the $\mathcal{L}(s_i)$ set for each vertex. These references are used as a look-up by the dominance check in

lines 23-29, where the algorithm needs fast access to all of the labels associated with a particular vertex.

In each **while-loop**, the algorithm checks whether the Fibonacci heap still contains elements. If this is not the case, there is no route which can be enlarged and, therefore, no route from s_i to t_i , starting at $time_i$, exists (line 32). If the Fibonacci heap still contains elements, the algorithm takes a minimal element with respect to the key (line 7), checks whether this label already represents a route to the target t_i (line 8-10) or, otherwise, tries to expand the associated route.

The route can usually continue along a number of different outgoing edges from any vertex and can potentially use different time-windows on each edge (line 11 and 12). In order to use an edge there must be a time-window available with an overlapping time interval, as expressed by the conditions in line 14 and 16. The earliest possible point in time that edge e_L can be left is identified (line 18 and 19) and the expansion step is executed. When the condition stated in line 20 is true, a new label will be generated (line 21 and 22). Different cases are possible at this stage. Firstly, the new label may dominate another label (line 27), in which case the dominated label will be erased (line 28 and 29). Secondly, the new label may be dominated by an older one (line 25), in which case it is not necessary to take this label into account (line 26). The **while-loop** is executed as long as there is a route which can be expanded. If a route R to the target t_i has been found, the route can be generated by working backwards through the set of labels (line 9) using the references, $pred_L$, to the previous labels.

The generalized vertex-based Dijkstra's algorithm which has been presented here is a close variant of that given by Stenzle in [21] and solves the problem in polynomial time (polynomial in the number of time-windows). The proof can be found in [21], where it is presented for the edge-based algorithm.

4.5 Modifications to the QPPTW algorithm for airport ground movement

Algorithm 1 is used for arriving aircraft as described above, since their goal is to clear the runway and reach the gate/stand as quickly as possible. In our model, departing aircraft aim to reach the runway at a given time and leave the gate/stand as late as possible in order to do so. This allows for more of the waiting time to be absorbed at the gate/stand when the engines are not running. The same algorithm is used for this purpose, computing the route backwards with the end time fixed instead of the start time, with only minor changes to reverse the time-related steps. Since the algorithm logic remains unchanged, this modified algorithm has not been presented here.

To attempt to speed up the execution time of the algorithm, we applied goal-oriented search [20] to the QPPTW algorithm. Two heuristic measures were used to estimate a lower bound for the rest of the partial route: the Euclidean distance was used to measure the linear distance to the target, and the time was estimated using Dijkstra's algorithm to compute the time which would be needed ignoring any interference from other aircraft. Unfortunately, neither approach resulted in a valuable speed-up when applied to this problem. This can possibly be explained by the fact that the graph representing the airport layout is sparse (having on average only a few outgoing edges for each vertex) and routes often start on the border of the graph (see Figure 1), so the number of expansions exploring non-promising areas of the airport is relatively small.

4.6 Readjustment of the time-windows

When an aircraft has been routed, the time-windows have to be readjusted according to the edge utilisation of the adopted route R , and the edges which conflict with these. We note that it is necessary to consider edge conflicts only during this stage and not during the routing process (Algorithm 1).

Algorithm 2 presents the pseudocode for the readjustment of the time-windows. The input consists of the weighted graph $G = (V, E)$, the set of conflicting edges $confl(e)$ for all $e \in E$, the set of sorted time-windows $\mathcal{F}(e)$ for all $e \in E$, and the route R which was found for the most recent aircraft to be routed. The output is the new sorted set of time-windows $\mathcal{F}(e)$, including the reservations of the new route R .

Algorithm 2: Readjustment of the time-windows

Input: Graph $G = (V, E)$ with weights w_e for all $e \in E$, the route R with reservations $[time_f^{in}, time_f^{out}]$ for all $f \in R$, the set of sorted time-windows $\mathcal{F}(e)$ for all $e \in E$ and the set of conflicting edges $confl(e)$ for all $e \in E$.

Output: Sorted set of time-windows $\mathcal{F}(e)$ including the reservations of the route R

```

1  foreach  $f \in R$  do
2      foreach  $e \in confl(f)$  do
3          foreach  $F_e^j = [a_e^j, b_e^j] \in \mathcal{F}(e)$  do
4              if  $time_f^{out} \leq a_e^j$  then
5                  goto 2 /* time-window is too late */
6              if  $time_f^{in} < b_e^j$  then
7                  /* otherwise time-window is too early */
8                  if  $time_f^{in} < a_e^j + w_e$  then
9                      if  $b_e^j - w_e < time_f^{out}$  then
10                         Remove  $F_e^j$  from  $\mathcal{F}(e)$ 
11                     else
12                         /* shorten start of time-window */
13                          $F_e^j = [time_f^{out}, b_e^j]$ 
14                 else
15                     if  $b_e^j - w_e < time_f^{out}$  then
16                         /* shorten end of time-window */
17                          $F_e^j = [a_e^j, time_f^{out}]$ 
18                     else
19                         /* split time-window */
20                          $F_e^j = [a_e^j, time_f^{in}]$ 
21                         Insert  $[time_f^{out}, b_e^j]$  into set  $\mathcal{F}(e)$ 

```

Basically, the algorithm determines which other edges are blocked for each edge of the route R (lines 1 and 2). All affected time-windows on these edges are adjusted (line 3-7) and four different cases then have to be considered, depending upon the relative

positions of the time-windows. The remaining time-window may be removed (line 9-10) if it becomes too short to allow an aircraft to taxi; be shortened at the start (line 11-13) or shortened at the end (line 15-17); or it could be split in two smaller windows (line 18-21).

Once a route has been allocated to an aircraft, some additional waiting times may be required on edges, beyond the time required to traverse the edge as specified by the time intervals on the labels by Algorithm 1. Time intervals on adjacent edges often overlap sufficiently that there is a choice of which edge the wait can be assigned to. In our implementation, the waiting times are forced to be as late in the corresponding part of the route as possible (apart from the initial waiting time for departures, which is maximised to maximise the stand hold). Alternative approaches could use this flexibility to select better and smoother speed profiles for the aircraft. Using a similar approach to that used in [14], the aim could be to spread the necessary waiting times for an aircraft in such a way that the speed profiles are as “engine friendly” as possible. Although the effects of such postprocessing are not studied within this paper, they are an area which we intend to investigate.

4.7 Buffer times

The solutions of the approach are conflict-free routings, but it is possible for small delays to affect the entire plan. Buffer times would allow small deviations from the taxi times to be absorbed. To achieve such buffer times the label intervals in the algorithm are lengthened in the desired direction (before or after) by a certain amount. To reflect growing uncertainties along the route, the amount of time can be made distance-dependent. Buffer times could also depend upon the expected congestion at the time, being increased when delays were expected to be more likely, although at these times the introduction of a buffer time would be more likely to reduce throughput. This is an interesting area for future study.

4.8 Initial sequencing of taxiing aircraft

Burgain et al. [8] presented the concept of using collaborative virtual queues, extending previous gate holding concepts. The idea was to limit the number of aircraft which were taxiing on the surface and add all which want to start taxiing to a virtual queue, forcing them to wait until the count allows other aircraft to pushback. One benefit of this approach for the airlines is that they can swap their own aircraft in the virtual queue according to their priority level. Based upon this concept, the aircraft are initially ordered for the sequential routing algorithm by either the expected wheel-on time on the runway or the expected pushback time at the gate/stand. However, there are potentially better sequences which could be used to sequentially route the aircraft and we intend to analyse this potential.

5 Results and discussions

This section starts with a table collating the results to ease comparison. The explanation of the results will follow. The results of the taxi time estimation which was

presented in Section 3 are then discussed. An analysis of the results from the ground movement decision support system, which was described in Section 4, is then provided and this section ends with a summary of the potential benefits of such a system.

Table 2 Summary of the results

	Total taxi time [s]	Average taxi time per aircraft [s]
Actual total taxi time	271851.6	400.4
Statistical approach		
Total taxi time estimation	266990.5	393.2
Total taxi time estimation (unimpeded)	203331.8	299.5
QPPTW algorithm		
Using unimpeded taxi time estimates	207722.5	305.9
Using impeded taxi time estimates	276927.9	407.8

The relevant results are summarised in Table 2. The first row of results shows the actual total and average taxi times for the supplied dataset, including queuing time at the runway. The taxi speed function which was developed was then applied to each aircraft, to estimate the taxi times and the results are shown in the next two rows. In the first case, the function was applied assuming the actual traffic level and we note that the difference between the predicted and actual times is less than 2%. In the second case, the traffic related components of the function were zeroed, to estimate the taxi times if there had been no delays due to other aircraft, and the difference illustrates the amount of the taxi time which was a result of such delays. The two sets of taxi times (excluding and including delays for other aircraft) were then used within the QPPTW algorithm and the total and mean resulting taxi times are shown in the table for each case. These results are analysed and explained further in the following two sections.

5.1 Analysis of taxi time estimation

The taxi speed analysis by Atkin et al. [5] was used for taxi time prediction within this research. The goal of the multiple regression model was to find the most important factors explaining the variability of the data in the real dataset. Extensive analysis was performed, seeking a function which was as practical as possible (requiring less information), easy to interpret, and maintained a high level of accuracy.

The estimation approach has two important uses. Firstly, the overall taxi time (including runway queue time) can be estimated, taking into account the other traffic at the airport. Secondly, the approach could be used to estimate the taxi time for an unimpeded aircraft. The coefficients which were found for each factor are specified in the paper by Atkin et al. [5], and the shortest route for each aircraft between the runway and gate/stand was used for the taxi distance/speed calculation. Unimpeded taxi times for aircraft can be used in our approach since the decision support tool takes the effects of the other aircraft into consideration itself, performing the necessary re-routing or delaying of aircraft.

The results from the statistical model can be seen in Table 2. The model implies that 23.8% of the taxi time was related to delays due to other aircraft, including delays in queues behind other aircraft at the runway. There would be an average saving of 93.7s per aircraft if these interactions could be eliminated. The influence of the interactions between the aircraft which lead to the waiting times is analysed in the next section.

5.2 Experimental details using the QPPTW algorithm

The framework was programmed in Java as a single-threaded application and executed on a personal computer (Intel Core 2 Duo, 3GHz, 2GB RAM). In these experiments, all aircraft were allowed to use all of the taxiways and only intersecting and adjacent edges were considered to be in conflict and were, therefore, not allowed to be used by two aircraft simultaneously. The buffer time (Section 4.7) was set to zero. Analysis of different buffer times showed that the taxi time would have been enlarged only by a linear factor of the buffer time. Similar results were also found in [18].

Extensive analysis was performed using the QPPTW algorithm to solve the ground movement problem for the layout and data from Zurich Airport. The aircraft were routed sequentially using the taxi time estimations from the statistical analysis of taxi times which was discussed in Section 3. The resulting total taxi times can be found in Table 2, where the taxi times used were those which were estimated for unimpeded aircraft (ignoring the influence of factors related to other aircraft on the surface), the average taxi time (including re-routing and waiting delays) was 305.9s per aircraft. In contrast, the average taxi time per aircraft for the impeded taxi time estimations was 407.8s, indicating that 25.0% (or 101.9s per aircraft) of the observed taxi time was due to runway holds and re-routing or queuing due to other aircraft.

The estimations of the taxi times from the statistical approach provide a lower bound for the taxi times, since they assume no re-routing delays or queuing behind other aircraft. The QPPTW algorithm is designed to predict the delays which are actually necessary due to the interactions between aircraft for the specific routings and timings which the algorithm assigns to aircraft. Comparison of the resulting taxi times from the QPPTW algorithm against the lower bound reveals an increase in the taxi time from 203331.8 to 207722.5 seconds, showing that the additional taxi times for the re-routing and waiting summed to 4390.7s over the entire day, an increase of around 2.2% in the total taxi time. The 2.2% increase over the lower bound (rather than optimal) times indicates that its use as a ground movement decision support system seems very promising for this problem.

It is also interesting to compare the approach described here against the actual performance of the airport on this particular day of operation. Data from Zurich Airport reports a total taxi time of 271851.6s. Comparison with the results for the QPPTW algorithm with unimpeded taxi time estimation highlights savings of about 23.6% or an average of 94.5s per aircraft. Obviously, this only indicates an upper bound for the potential savings, since the real times will include some slack time for the departures at the runway to ensure a high runway throughput.

The solution time to solve the entire day of operation with 679 aircraft was 11609ms. This correlates to an average solution time of 17ms per aircraft, which supports the potential use of the algorithm in an online decision support system. No infeasible solution occurred within any of the executions of the simulation.

6 Conclusion

This paper described a more realistic and potentially environmentally friendly ground movement decision support system, compared to previous approaches. The framework is designed to combine the runway sequencing problem and ground movement problem, aiming for better global solutions, although only the ground movement element was considered in this paper. This work extends the basic ground movement problem of minimising the travel times to include the concept of absorbing possible waiting times for departures at the gate/stand, to reduce the fuel burn and environmental impact. The sequential QPPTW algorithm which was described here is based on graph theoretical concepts and can include restrictions such as limitations of which taxiways aircraft can use, which taxiways block which and when, and turn limitations at taxiway junctions. In addition, the algorithm provides the opportunity to add buffer times for blocking the reserved taxiways for longer than expected, to absorb small delays and schedule disturbances.

Data was used from an entire day of operations at Zurich Airport, the largest hub airport in Switzerland. This data was used to generate more accurate taxi time estimations for each aircraft, using a taxi speed prediction function which was generated from an extensive statistical analysis of the same dataset. These taxi time estimations were then utilised within the QPPTW algorithm to route and schedule the ground movement. The results are very promising and show potential maximum savings in total taxi time of about 23.6% from using the decision support system described here, together with the statistical taxi time prediction system, compared to the actual performance at the airport. Obviously, further research is necessary to determine the amount of buffer time and runway delay which should be utilised to account for any remaining taxi time uncertainty to avoid starving the runways.

The experimental results of the developed decision support approach show average solution times of only a few milliseconds per aircraft, and are, therefore, adequate for the implementation of such a system for real time use at airports.

We intend to investigate various extensions of this work in future, in addition to the combination of the ground movement problem with the runway sequencing problem. Firstly, the QPPTW algorithm enables the possible waiting times to be spread in different ways. In this paper, they were allocated so as to maximise the stand hold time and to better adapt to schedule disturbances, but an alternative approach would be to develop smoother speed profiles for aircraft, using the engine in a more efficient and environmentally friendly way. Secondly, there are potential benefits to be gained from improving the sequence of consideration of aircraft for the sequential routing algorithm. Finally, we would like to perform a similar analysis for different airport layouts, to better understand the effects of the layout upon the best solution approach, but it will be necessary to obtain more data and support from other airports in order to do so.

Acknowledgements The authors wish to thank the Engineering and Physical Sciences Research Council (EPSRC) for providing the funding which made this research possible. We would also like to thank Flughafen Zürich AG who provided the real dataset.

References

1. ACI EUROPE: An outlook for Europe's airports - Facing the challenges of the 21st century. Tech. rep., Airports Council International Europe (2010). <http://www.aci-europe.org/policyoutlook.pdf>
2. Atkin, J.A.D., Burke, E.K., Greenwood, J.S.: TSAT allocation at London Heathrow: The relationship between slot compliance, throughput and equity. *Public Transport* **2**(3), 173–198 (2010). DOI 10.1007/s12469-010-0029-2
3. Atkin, J.A.D., Burke, E.K., Greenwood, J.S.: A comparison of two methods for reducing take-off delay at London Heathrow airport. *Journal of Scheduling* (2011). DOI 10.1007/s10951-011-0228-y
4. Atkin, J.A.D., Burke, E.K., Greenwood, J.S., Reeson, D.: Hybrid metaheuristics to aid runway scheduling at London Heathrow Airport. *Transportation Science* **41**(1), 90–106 (2007). DOI 10.1287/trsc.1060.0163
5. Atkin, J.A.D., Burke, E.K., Maathuis, M.H., Ravizza, S.: A combined statistical approach and ground movement model for improving taxi time estimations at airports. Submitted.
6. Atkin, J.A.D., Burke, E.K., Ravizza, S.: The airport ground movement problem: Past and current research and future directions. In: *Proceedings of the 4th International Conference on Research in Air Transportation (ICRAT)*, Budapest, Hungary (2010)
7. Balakrishnan, H., Jung, Y.: A framework for coordinated surface operations planning at Dallas-Fort Worth International Airport. In: *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Hilton Head, SC, USA (2007)
8. Burgain, P., Feron, E., Clarke, J.P.: Collaborative virtual queue: Benefit analysis of a collaborative decision making concept applied to congested airport departure operations. *Air Traffic Control Quarterly* **17**(2), 195–222 (2009)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms*, 2nd edn. MIT Press and McGraw-Hill (2001)
10. Dijkstra, E.W.: A note on two problems in connexion with graphs. *Numerische Mathematik* **1**, 269–271 (1959)
11. Dorndorf, U., Drexler, A., Nikulin, Y., Pesch, E.: Flight gate scheduling: State-of-the-art and recent developments. *Omega* **35**(3), 326–334 (2007)
12. Gawrilow, E., Köhler, E., Möhring, R., Stenzel, B.: Dynamic routing of automated guided vehicles in real-time. In: *Mathematics - key technology for the future*, pp. 165–178. Springer (2008)
13. Idris, H.R., Clarke, J.P., Bhuva, R., Kang, L.: Queuing model for taxi-out time estimation. *Air Traffic Control Quarterly* **10**(1), 1–22 (2002)
14. Lesire, C.: Iterative planning of airport ground movements. In: *Proceedings of the 4th International Conference on Research in Air Transportation (ICRAT)*, Budapest, Hungary (2010)
15. Marín, Á.: Airport management: Taxi planning. *Annals of Operations Research* **143**(1), 191–202 (2006)
16. Marín, Á., Codina, E.: Network design: Taxi planning. *Annals of Operations Research* **157**(1), 135–151 (2008). DOI 10.1007/s10479-007-0194-0
17. Rappaport, D.B., Yu, P., Griffin, K., Daviau, C.: Quantitative analysis of uncertainty in airport surface operations. In: *Proceedings of the AIAA Aviation Technology, Integration, and Operations Conference* (2009)
18. Ravizza, S.: Control of automated guided vehicles (AGVs). Master's thesis, ETH Zurich, Switzerland (2009)
19. Roling, P.C., Visser, H.G.: Optimal airport surface traffic planning using mixed-integer linear programming. *International Journal of Aerospace Engineering* **2008**(1), 1–11 (2008). DOI doi:10.1155/2008/732828
20. Sedgewick, R., Vitter, J.S.: Shortest paths in euclidean graphs. *Algorithmica* **1**(1), 31–48 (1986). DOI 10.1007/BF01840435
21. Stenzel, B.: Online disjoint vehicle routing with application to AGV routing. Phd thesis, TU Berlin, Germany (2008)

A Game Theoretic Approach for the Taxi Scheduling Problem with Street Hailing

Ruibin Bai, Jiawei Li, Jason A D Atkin, Graham Kendall

Abstract This paper describes a taxi scheduling system to improve the efficiency of taxi services which are mainly implemented by means of street hailing. This is of particular relevance to Chinese cities, where this is by far the most common way in which taxis are requested, since the majority of taxi drivers are independent rather than being affiliated with a specific company. The mobile phone and GPS-based taxi scheduling system which is described in this paper aims to provide a decision support system for taxi drivers and facilitates information exchange between taxi drivers and passengers, while allowing drivers to remain independent. The taxi scheduling problem is considered to be a non-cooperative game between taxi drivers and a description of this problem is given in this paper. We adopt an efficient algorithm to discover a Nash equilibrium, such that each taxi driver and passenger cannot benefit from changing their assigned partner. Two computational examples are given to illustrate the effectiveness of the approach.

1 Introduction

Taxis play an important role in modern public transportation networks, especially in countries such as China and India, where bus and underground train services are still under development and do not currently have the coverage required by passengers. Taking a taxi is a convenient way to get to your destination, although it is usually slightly more expensive than buses and subways. According to a 2006 survey, there are approximately one million taxis in service around mainland China and the two million people who are involved in the taxi industry turnover more than 29.5 billion US dollars annually [1].

We can broadly classify taxi services into three categories: street hailing, taxi-stands and prearranged bookings [21]. Street hailing is a common method for getting a taxi, especially in small cities where the taxi service is still at a relatively small scale. Street hailing is simple and has no requirement for expensive equipment or systems. However, as will be analyzed later in this section, street hailing causes several problems, including low taxi utilization, long passenger waiting times, road safety issues, and traffic congestion. Taxi-stands have fewer of these problems but can only serve a small group of passengers at fixed locations, for example,

Ruibin Bai

Division of Computer Science, University of Nottingham Ningbo, China, Ningbo, 315100, China.

E-mail: ruibin.bai@nottingham.edu.cn

Jiawei Li

School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK

E-mail: jwl@cs.nott.ac.uk

Jason A D Atkin

School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK

E-mail: jaa@cs.nott.ac.uk

Graham Kendall

School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK

E-mail: gxk@cs.nott.ac.uk

city centers, bus/train stations, airports, etc. Prearranged booking is widely considered to be a more advanced and efficient taxi dispatching method, but is impractical when there are many independent taxi drivers. The booking can be performed via traditional telephone calls or by utilizing other modern communication technologies (such as websites or text messages). A taxi dispatching center is often employed to collect all the passenger information and coordinate the taxi dispatching centrally. The dispatching systems which are used often utilize the latest GPS devices and wireless communication equipment. The advantage of such a pre-booking system is that the demands of passengers are clear and known in advance. One drawback is that the system is often expensive and, therefore, suitable only for large taxi companies. In addition, the system relies upon mutual trust between taxi drivers and passengers, that neither party will break the agreed appointment. Unfortunately, in reality, such breaking of appointments happens frequently in many countries.

Unlike in the EU and USA, prearranged booking is not a popular method for hailing taxis in China. Most taxi drivers are actually self-employed, although they may be registered as employees for some taxi companies. It is hard to build a prearranged booking mechanism where taxi drivers who are self-employed (or employed by many different taxi companies) would work cooperatively. An efficient communication network would also be needed to implement this type of taxi scheduling, especially in a city with a large population. In China, a very limited number of taxi companies provide a prearranged booking service, except in some large cities such as Beijing and Shanghai. Therefore, street hailing is still the most frequent method of getting a taxi in China and approximately 85% of taxi services are implemented in this way. However, as mentioned earlier, street hailing is inefficient and poses both safety and congestion problems. Taxi drivers are unable to predict exactly when and where to meet the next passenger and passengers cannot be sure how long they will have to wait for the arrival of a taxi, leading to high rate of wasted mileage for taxi drivers and longer waiting times for passengers. In addition, taxi drivers need to concentrate on both driving and finding the next passenger, increasing the risk of accidents. Frequently, too many taxis crowd the city centers and passenger hotspots, leading to congestion problems.

There have been several research projects and technological innovations for pre-arranged booking systems, often abstracted and formulated as the Dial-A-Ride (DAR) problem, which falls into the category of classic Vehicle Routing Problems (VRP), or the Traveling Repair Problem (TRP) [6]. Several on-line algorithms have been developed, using linear programming [7], heuristics [8-10] and meta-heuristics [11]. Global positioning systems (GPS) and Geographic information systems (GIS) have been used by vehicle dispatching systems in many cities, to improve the efficiency of the taxi service [2-5]. For example, a GPS-based Automatic Vehicle Location and Dispatching System (AVLDS) greatly improved the quality and efficiency of taxi services in Singapore. Once a request for a taxi is received, the control center will send signals to the taxis which are located within 10km of the passenger. If one of the drivers accepts the job, he/she can respond to the control center by merely pressing a button which is installed in his/her taxi. Once the job has been assigned, information about the taxi number and expected time of arrival will be immediately sent to the passenger [2, 3].

Research and innovation to improve the street hailing method has been very limited. The most relevant research that we are aware of includes models of the taxi market to analyze the demand and supply equilibrium [12, 13], the bilateral searching and meeting between passengers and taxis on a road network [14], and the taxi cruising policy, to improve the efficiency of the taxi service [15]. However, these models do not consider scheduling algorithms for street hailing. The lack of research in this field is probably due to the fact that most taxis that are involved in street hailing are self-employed and independent. Unlike in a centralized system, coordination and cooperation between these taxis is very difficult, or even impossible, without a good mechanism. The systems and dispatching methods which are used in the pre-arranged booking method are simply not suitable for street hailing. The independent nature of the taxi drivers using such a system cannot be ignored, and perhaps the most important consideration for such a system is ensuring that it allocates passengers to taxis so that it is always in the best interests of the taxi drivers to keep the passengers they have been

allocated. The focus of this paper is upon a novel system for helping with the street hailing problem, to complement existing taxi booking systems.

In the remaining sections, we first describe the street hailing problem that we want to address. Section 3 introduces a novel game theoretic model of street hailing, along with a solution method for the problem. We then adopt the proposed approach and provide two computational examples in Section 4. Finally, a brief summary of the conclusions of this study and the directions for future work are presented in Section 5.

2 Street Hailing Using a New System

We propose to use the following system to improve the taxi street hailing process. The system adopts the latest GPS, GIS, and 3G/GPRS wireless network and comprises a central server and independent client applications for the passengers and taxis (see Figure 1). The server and clients are connected via a 3G/GPRS network. In our implementation, the clients are smart phones with GPS and GIS support and automatically send location and status (occupied or vacant) information to the server via the 3G/GPRS network. The server periodically analyses the geographical distribution of clients on a digital map, generates a stable taxi-passenger scheduling and then broadcasts the allocations back to clients.

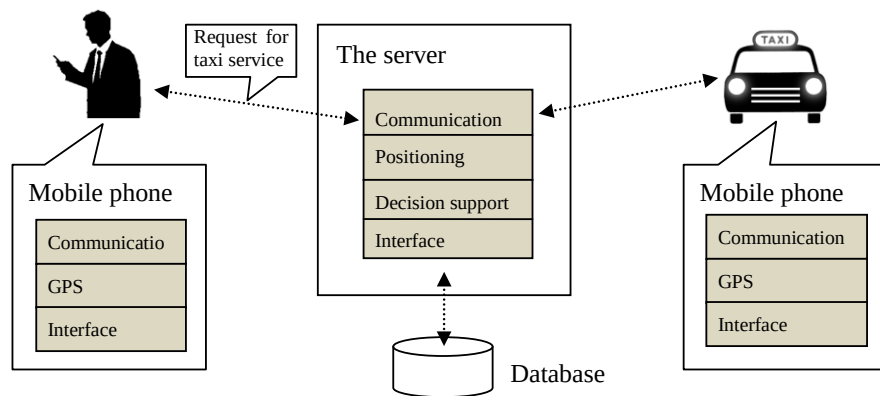


Figure 1 Taxi scheduling system

In practice, there may be several ways to perform the taxi scheduling under this new system. A straightforward approach would be providing passengers and taxis with a two-way communication channel to negotiate a manual schedule, although this may not be efficient. An improved design would provide automatic decision support for the taxis and passengers by taking full advantage of the geographical distribution information of the passengers and taxis on a digital map. This paper focuses on the latter implementation and proposes a scheduling procedure that is illustrated in Figure 2. As can be seen from the figure, the city map is divided into smaller zones to facilitate better taxi management and unnecessary complexity. A schedule is generated periodically (defined by the parameter t_w) rather than in real-time. To avoid confusion, the following assumptions are also made about the problem under consideration in this paper:

1. Taxis are independent and compete against each other in order to gain maximum personal return (i.e. there is no subset of taxis that plays a cooperative game).
2. A taxi's primary goal is to find the next passenger within the shortest possible time.
3. Scheduling recommendations by the system are only supportive, not compulsory. Therefore, a taxi can decline a recommendation by the system in certain circumstances (e.g. if the recommended passenger is too far away, although the division of the city into zones should reduce these occurrences).

4. Locations of passengers and taxis are automatically retrieved from the embedded GPS chip and sent to the server by the software client.
5. A passenger's destination information is not known when a schedule is generated. Therefore, this system cannot plan a taxi's route beyond its very next service. This means the system is not a pre-booking taxi system but serves as a supplement system for the latter.

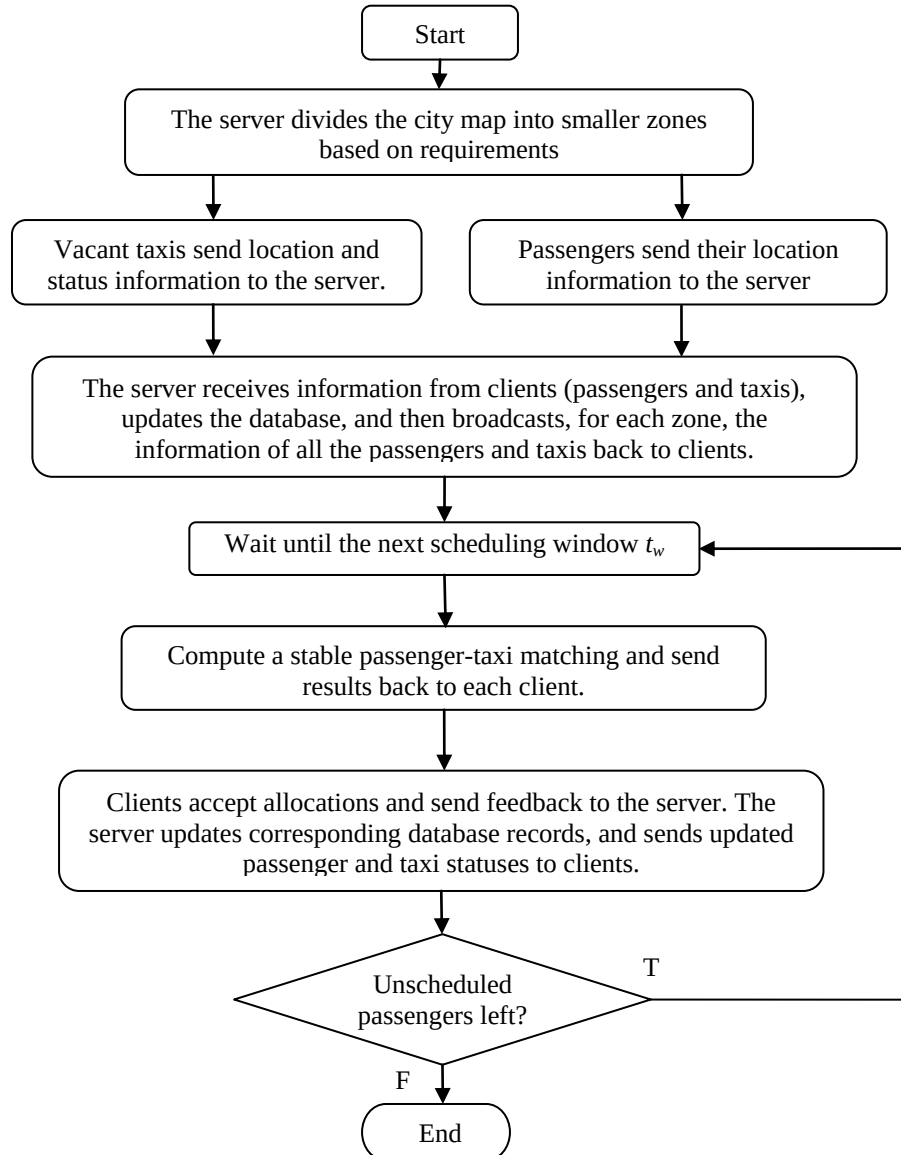


Figure 2. Scheduling process

We note that some of these assumptions may be relaxed in practical implementations. The main purpose of their inclusion in this paper is to introduce a basic problem that can be studied initially to gain some insightful knowledge, and then be extended later. We acknowledge that the research reported in this paper is still in the early stages as far as fully addressing the practical problems and that several challenging but interesting practical problems have arisen. We believe that the research results reported in this paper, as well as the new problems raised

in this paper, would be of interest for other researchers and could potentially lead to increased research attention for this important research area.

3 A Game Theoretic Model

In contrast to the usual algorithms for the DAR problem, we develop a game theoretic model that treats the taxi scheduling problem as a non-cooperative game between taxi drivers. The taxi drivers are assumed to be self-interested and to compete rather than cooperate with each other in serving passengers. Given the locations of passengers and other taxis, each taxi driver will have preferences for some passengers over others and each passenger may also have preferences for taxis (e.g. a preference to be picked up as quickly as possible). We aim to provide an algorithm to generate a matching of taxis and passengers that will be accepted by all taxis.

A matching of taxis and passengers is a Nash equilibrium if no taxi or passenger can find a better partner than their current one, so that nobody has any incentive to deviate from the suggested allocation. In other words, any alternative taxi a passenger may wish to switch to would prefer its currently allocated passenger and any alternative passenger which a taxi may wish to switch to would prefer its currently allocated passenger, so no unilateral decision by a taxi or passenger will result in a benefit for that taxi or passenger. Such an allocation would then be appropriate for utilization in a system to support street hailing.

Let T_i ($i=1,...,n$) denote the taxis, P_j ($j=1,...,m$) the passengers and t_{ij} time for taxi i to reach passenger j . Let E_{ij} represent the preference of taxi driver i for passenger j . We consider a computerized street-hailing system in this paper rather than a pre-booking system and we assume that the taxi drivers are not given information about the destination for any passenger. Taxi drivers can, therefore, only utilize the t_{ij} information for their passenger preferences and will prefer passengers which are closer to them, as previously discussed. We therefore assume that E_{ij} is entirely determined by t_{ij} and is negatively correlated with t_{ij} . For the experiments for this paper, we assume that E_{ij} is defined as follows, in order to meet these requirements:

$$E_{ij} = \frac{1}{t_{ij}}, \quad i=1,...,n; j=1,...,m \quad (1)$$

The ability of taxi A to take a passenger off another taxi B to which it has been assigned relies upon both the passenger being willing and taxi A being able to reach the passenger before taxi B . We assume that passengers have no preference for taxis other than that they wish to be picked up as early as possible, so these two factors can be considered to be identical in this case. A passenger preference, representing the ability of a taxi driver to take this passenger from another taxi driver, could then be determined by assuming that the preference of passenger P_j for taxi T_i is negatively correlated with t_{ij} . Taxi A can then take passenger j away from another taxi B if and only if $t_{Aj} < t_{Bj}$, i.e. if it can reach the passenger first.

If we consider the taxi drivers' preferences (E_{ij}) as the scoring of one partner and the aforementioned passenger preference as the scoring of the other partner then the taxi-passenger pairing problem can be considered to be a variant of the stable marriage problem. The stable marriage problem [17,18] involves n men and n women, each of whom has ranked all members of the opposite sex with a unique number between 1 and n , in order of preference. The objective is to find a set of man-woman pairs such that there are no two people of the opposite sex who would both rather have each other than their current partners. The Gale-Shapley algorithm [16] was developed to solve this problem in polynomial time. This algorithm involves a number of 'rounds'. In each round, each unengaged man proposes to the most preferred woman to whom he has not yet proposed. Each woman then considers all proposals to her and chooses the one she most prefers as her partner and becomes engaged to him, potentially 'trading up' from an existing engagement. This process continues until there is no unengaged man. The Gale-Shapley algorithm guarantees a stable matching.

The Gale-Shapley algorithm is still applicable when there are an unequal number of partners, i.e. $n \neq m$, so can be applied to the taxi-passenger matching problem. The algorithm is

either man-oriented or woman-oriented. That is, each man (or woman) is paired with the best partner that they can have in any stable matching. In the case of $n \neq m$, a stable matching is guaranteed, but some men (or women) will not be paired.

For the taxi scheduling problem considered here, given the definition of $\{E_{ij}\}$ and the consequent correlation between the two rankings, the Gale-Shapley algorithm can be simplified. We have adopted an algorithm which first sorts all elements in the preference matrix $\{E_{ij}\}$ into decreasing order, then considers each item in turn, from the first to the last. For each element E_{ij} , if neither taxi i nor passenger j has been allocated then i is allocated to j and (i, j) becomes part of the stable matching. The time complexity of the proposed algorithm is $O(mn \times \log(mn))$ (since the sorting is the slowest element). It is easy to verify that the stable matching is a Nash equilibrium, since the only passengers that a taxi driver could beneficially switch to will have been allocated to a taxi which is even closer to them.

There must be at least one stable matching (Nash equilibrium), given the preference matrix $\{E_{ij}\}$, however, it could be difficult to find the Nash equilibria since the search space of potential combinations of rankings is so large. The Gale Shapley algorithm will find one of the equilibria, but it is probable that there is more than one stable matching and the others may be much harder to find. In game theory, the price of anarchy (PoA) is used to describe the ratio between the maximum social payoff and the payoff from an equilibrium [19]. We use it here to denote the difference between the total preference value of the stable matching which we have found and the preference value of the optimal stable matching. As an example, Figure 3 shows example locations for two taxis and two passengers, showing the selected and possible pairings. In Figure 2(a), both $\{T_1, P_2\}\{T_2, P_1\}$ and $\{T_1, P_1\}\{T_2, P_2\}$ are equilibria because T_1 is indifferent between choosing P_1 and P_2 , but is closer to both passengers than T_2 is. However, $\{T_1, P_2\}\{T_2, P_1\}$ has a better overall preference, thus a lower PoA. In Figure 2(b), both taxis T_1 and T_2 prefer P_2 to P_1 and there are two equilibria. The equilibrium $\{T_1, P_2\}\{T_2, P_1\}$ has a higher total preference, so a lower PoA.

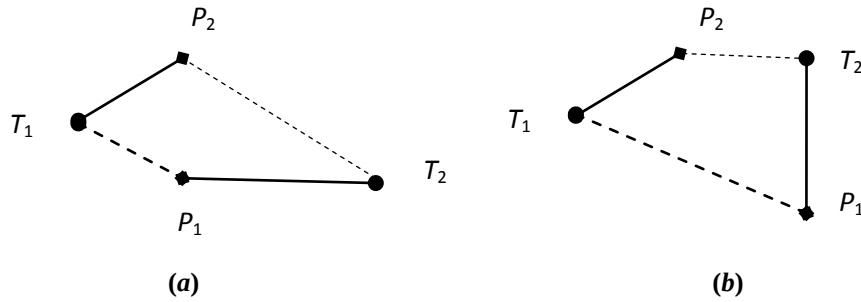


Figure 3 There are multiple equilibria because (a) taxi T_1 is indifferent between choosing P_1 and P_2 and (b) both T_1 and T_2 prefer choosing P_2 to P_1 . The equilibrium with lower PoA is $\{T_1, P_2\}\{T_2, P_1\}$ in both cases.

In order to minimize the PoA, a system may need to compute all Nash equilibria, to find the equilibrium with the highest total preference, but this would lead to an unacceptable amount of computation when the numbers of taxis and passengers are large. However, we have found that the differences between the total preference values for the equilibria are actually very limited in the taxi scheduling problem, as we will show in section 4.1.

4 Computational Examples

In this section, we simulate the taxi scheduling problem in a square area 50km x 50km. Taxi drivers and passengers are most likely to appear around hotspots such as business centers, airports, train stations and so on. The locations of passengers were approximately distributed around these hotspots according to the Poisson distribution [20, 21]. The distance between a

taxi and a passenger was determined by the Euclidean distance between their locations. Distances were considered to an accuracy of 0.15km, so that two distances were considered to be same if their difference was less than 0.15km.

4.1 Optimal matching

In this simulation, a taxi scheduling problem of 1,000 taxis and 1,000 passengers was solved using the simplified Gale-Shapley algorithm described earlier. The locations of taxis and passengers are shown in Figure 4. There were three hotspots.

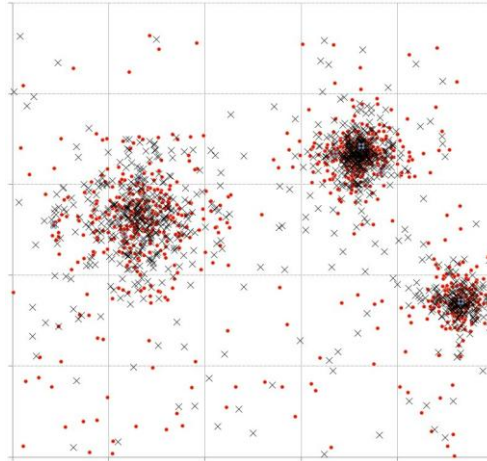
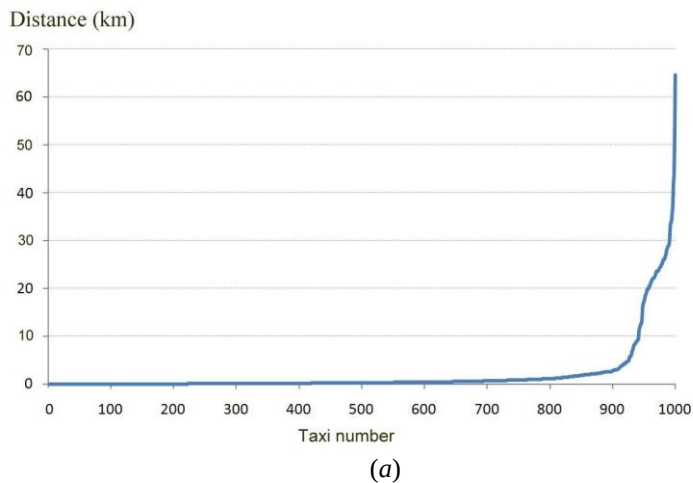
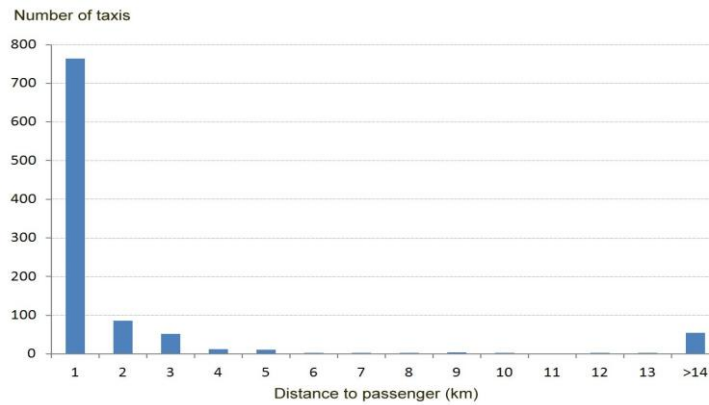


Figure 4 Distribution of 1,000 taxis and 1,000 passengers in a square area (taxis are denoted by x marks and passengers are denoted by dots).

The optimal matching of 1,000 pairs of taxi and passenger was also computed and the distances between the taxis and the passengers are shown in Figure 5(a), ordered from smallest to largest. Figure 5(b) shows the number of taxis which fall into different distance ranges. It shows that most of the scheduled taxis travel only short distances to their passengers, while a small number of taxis need to travel long distances (more than 14km).





(b)

Figure 5 Optimal matching of 1,000 taxis and 1,000 passengers. (a) Distances to passenger of 1,000 taxis. (b) Statistic of number of taxis corresponding to travel distance.

We note that this scheduling system only provides decision support for taxi drivers. Taxi drivers will not necessarily choose to collect an assigned passenger or use the assigned route. In practice, a taxi should not be assigned a passenger who is very far away because doing so leads to a high, non-profitable mileage for the taxi driver and an intolerable waiting time for the passenger. Experiments were executed to compare the optimal solutions and the stable equilibria which the previously described algorithm obtained. We observed that, if we ignore the taxis that are assigned to distant passengers, there was little difference between the optimal matching and the stable equilibria. An example of such a comparison is given in Figure 6. The reason is that most taxis and passengers are located around the hot spots and the resulting short distances will be prioritized by the pairing method we use to achieve a stable matching. The amount of computation could be greatly reduced if we do not need to find the optimal matching and these results imply that this is practical.

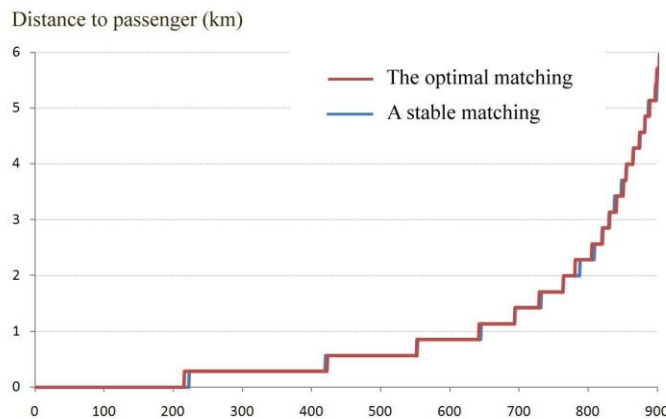


Figure 6 A comparison between the optimal matching and a non-optimal stable matching.

Although taxis which are assigned distant passengers would not necessarily pick up the passengers, the system could still be of use to these taxi drivers because it provides information about the distribution of passengers and other taxis. It can, therefore, help the taxi driver to choose where to go next to find the next passenger.

For comparison, a FCFS strategy was developed where the passengers were allocated to taxis one at a time according to the time sequence of their requests, and each passenger was assigned to the nearest taxi which was available. FCFS is a ‘fair’ algorithm for passengers but it increases the average vacant mileage for taxi drivers compared with the optimal matching. We show the comparison of the first come first serve (FCFS) strategy against the optimal matching in Figure 6.

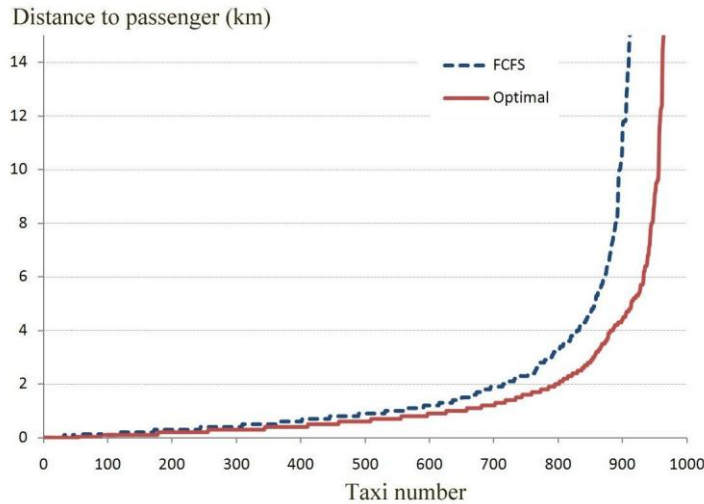


Figure 7 Comparison of the optimal matching with the result of FCFS strategy (the dashed line denotes the result of the FCFS strategy).

4.2 Algorithms for dynamic scheduling

In practice, taxi scheduling is a dynamic process and the time window for the request of taxi service needs to be taken into consideration. We simulated a scheduling problem of 5,000 taxis and 50,000 passengers and present the results here. The locations of taxis, passengers, and their destinations were within a square area. Travel times were calculated based upon the distance between the taxi and customer positions. Of course, taxi drivers are able to reject passengers (for example if a passenger was too far away) and would normally do so straight away in that case. The travel times for taxis (and the waiting times for serviced passengers) would then be lower but there would be un-serviced passengers. We assume for these experiments that taxi drivers do not reject passengers and therefore some passengers have relatively large waiting times. In these experiments, the requests for taxi service from the passengers were randomly distributed around rush hour according to the Poisson distribution, in addition to the random distributions of their locations and destinations. Three strategies were studied:

- First come first serve (FCFS). The nearest vacant taxi will be assigned immediately when the request for service is received.
- Strategy 1: The scheduling is performed every t_w minutes, where t_w is a parameter specifying a service window size. For these experiments, t_w was set to 5 minutes. The passengers whose requests for service are received within the 5 minute window are all scheduled together using the algorithm described in Section 3.
- Strategy 2: A combination of the FCFS strategy and strategy 1. A passenger will be immediately served according to the FCFS strategy if there is a vacant taxi within 10km distance (the 10km being another adjustable algorithm parameter), otherwise, strategy 1 will be adopted and a taxi will be allocated at the end of the current service window.

The results of this simulation are shown in Table 1. With strategy 1, there is a delay between the passenger requesting service and the service being scheduled (of up to 5 minutes

in the worst case). Although the average vacant mileage is decreased for the taxis compared with FCFS, the saving in passenger waiting time is counteracted by the scheduling delay. Strategy 2 is better than the other strategies in terms of its low waiting time of passengers.

Table 1 Result of simulation

Strategy	Taxis		Passengers
	Average mileage	Average vacant mileage*	Average waiting time**
FCFS	48.28	9.70	18.5
Strategy 1	52.01	6.45	19.1
Strategy 2	51.79	6.75	14.3

* Vacant mileage: travel distance for a taxi to reach the location of passengers.

** Waiting time: the time interval between receiving of the request of service and arrival of taxi.

The waiting times of passengers, mileages of taxis, and vacant mileages of taxis are shown in Figures 8-10 respectively.

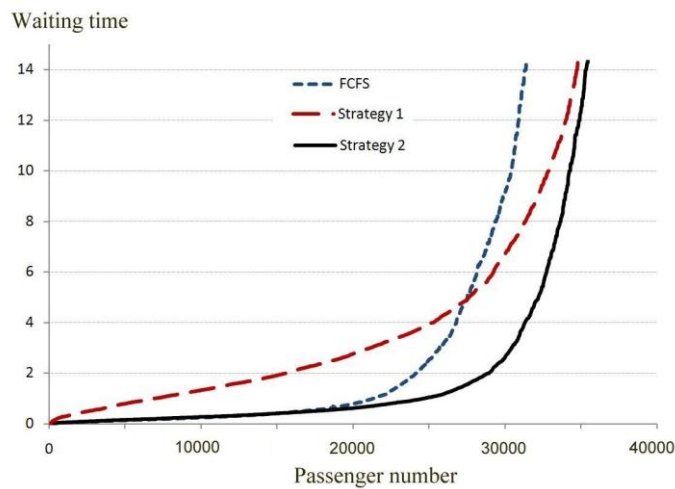


Figure 8 Waiting times for passengers, (the passengers with waiting time more than 14 minutes are not shown).

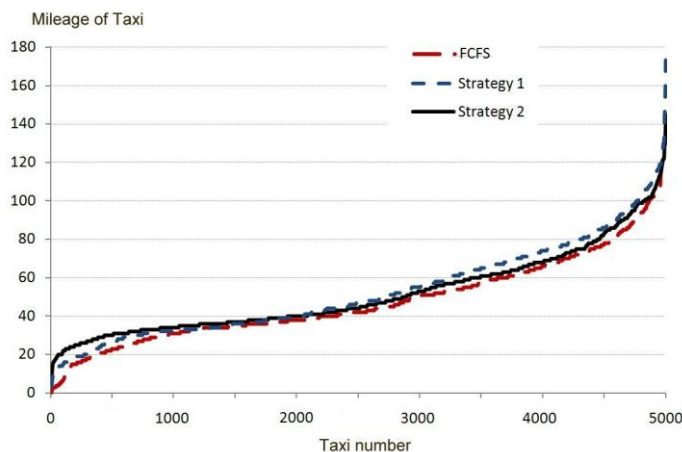


Figure 9 Mileages for the 5,000 taxis.

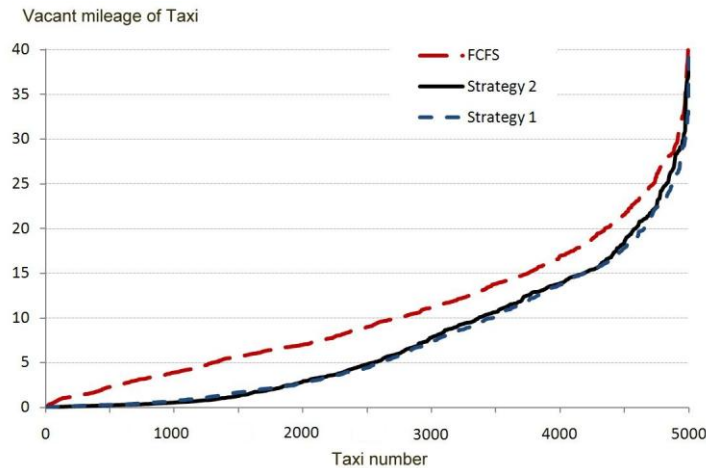


Figure 10 Vacant mileages for the 5,000 taxis.

The results of the simulation show that the proposed scheduling algorithm benefits both taxi drivers and passengers. It increases valid mileages and decreases vacant mileages for taxis. Low vacant mileages lead to low waiting times for passengers.

5 Conclusion and future research

Although this is only preliminary research into a scheduling algorithm for street hailing, we have described a low cost taxi scheduling system to improve the efficiency of taxi services. This would be particularly useful in Chinese cities, where the service is mainly implemented by means of street hailing. Based on mobile phones with GPS functionality, the system collects information about the locations of passengers and the locations and statuses of taxis and provides decision support for both taxi drivers and passengers. The simulation results in this paper show that our system could greatly improve the efficiency of street hailing.

The scheduling algorithm is based on a game theoretic model that considers the problem as a non-cooperative game between taxi drivers. An algorithm has been proposed which will discover a Nash equilibrium for this game, such that each taxi driver cannot find a better choice than their assigned passenger and route. The results of simulation show that this algorithm is efficient in computing the matching of taxis and passengers and it only needs small amount of computations. Thus, it is suitable for online taxi scheduling.

In addition to the time to pick up passengers, some other factors may influence a taxi driver's preferences, such as individual experience about traffic conditions or favoritism for a specific area. It is worth taking these subjective factors into consideration, where possible, in implementing a practical system, although doing so will increase the complexity of the algorithm.

There is always uncertainty in the real world taxi scheduling problem. A passenger may change their plans or lose patience in waiting, or a taxi may be late because of unexpected traffic congestion. The influence of uncertainty on the proposed scheduling algorithm and its robustness in a real world environment in the presence of uncertainty will form part of our future research into this interesting and important problem.

References

- [1] Yearbook of China Transportation & communications 2006, ISSN 1002-8617.
- [2] Z. Liao (2003) Real-time taxi dispatching using global positioning systems, *Communication of the ACM*, 46(5): 81-83.

- [3] D. Lee, Wang H. and Cheu R. (2003) Trip-chaining for taxi advance bookings: a strategy to reduce cost of taxi operation, *Proceedings of the 83rd Annual Meeting of the Transportation Research Board*,
- [4] S. Prakash and M. Kulkarni (2003) Fleet management: A GPS-GIS integrated approach, in *GIS Development*.
- [5] A. Glaschenko, A. Ivaschenko, G. Rzevski, and P. Skobelev (2009) Multi-agent real time scheduling system for taxi companies, *Proc. Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pp.29-36.
- [6] J. Cordeau and G. Laporte (2007) The dial-a-ride problem: models and algorithms, *Annals of Operations Research*, 153(1): 29-46.
- [7] C. Deng, H. Ong, B. Ang, and T. Goh (1992) A modeling study of a taxi service operation, *International Journal of Operations & Production Management*, 12(11): 65-78.
- [8] O. Madsen, H. Ravn, and J. Rygaard (1995) A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives, *Annals of Operations Research*, 60(1): 193-208.
- [9] S. Irani, X. Lu, and A. Regan (2004) On-line algorithms for the dynamic traveling repair problem, *Journal of Scheduling*, 7: 243-258.
- [10] Y. Luo and P. Schonfeld (2007) A rejected-reinsertion heuristic for the static dial-a-ride problem, *Transportation Research Part B: Methodological*, 41(7): 736-755.
- [11] S. Nkoane (2009) Operations research modeling of the taxi transport between Polokwane and Mankweng in the Limpopo Province South Africa, M.Sc. Thesis, University of Limpopo.
- [12] R. Cairns and C. Liston-Heyes (1996) Competition and regulation in the taxi industry, *Journal of Public Economics*, 59(1): 1-15.
- [13] D. Flores-Guri (2003) An economic analysis of regulated taxicab markets, *Review of Industrial Organization*, 23(3-4): 255-266.
- [14] H. Yang, C. Leung, S. Wong, and M. Bell (2010) Equilibria of bilateral taxi-customer searching and meeting on network, *Transportation Research Part B*, 44: 1067-1083.
- [15] D. Flores-Guri (2005) Local exclusive cruising regulation and efficiency in taxicab markets, *Journal of Transport Economics and Policy*, 39(2): 155-166.
- [16] D. Gale and L. S. Shapley (1962) College Admissions and the Stability of Marriage, *American Mathematical Monthly*, 69: 9-14.
- [17] D. Gusfield and R. Irving (1989) The stable marriage problem, ISBN-10: 0-262-07118-5.
- [18] M. Halldorsson, R. Irving, K. Iwama, D. Manlove, S. Miyazaki, Y. Morita, and S. Scott (2003) Approximability results for stable marriage problems with ties, *Theoretical Computer Science*, 306(1-3): 431-447.
- [19] J. Corbo and Parkes D. (2005) The Price of Selfish Behavior in Bilateral Network Formation, in *ACM Symposium on Principles of Distributed Computing*, pp. 99-107.
- [20] W. Bailey and T. Clark (1987) A simulation analysis of demand and fleet size effects on taxicab service rates, *Proceedings of the 19th Conference on Winter Simulation (WSC'87)*, pp. 838-844.
- [21] S. Li (2006) Multi-attribute taxi logistics optimization. M.Sc. Thesis, Massachusetts Institute of Technology.

A Two-Campus Transport Problem

Elad Shufan · Hagai Ilani · Tal Grinshpoun

Abstract The presented two-campus transport problem (TCTP) is a pickup-and-delivery scheduling problem. It is a problem of a college with two campuses located at two different cities. Lecturers living in one city should sometimes teach at the other city campus. The problem is that of transporting N lecturers from one campus to the other, using a fleet of k vehicles, so as to minimize the time the lecturers wait for their transport. This is an optimal set partition problem. We mathematically model the one-directional case of the TCTP. The feasible space solution is represented by linear extensions of the requested departure time partially ordered set and reduction to the shortest path problem. This enables finding the optimal solution for the one directional case. TCTP is a two-destinations version of a dial-a-ride problem (DARP).

1 Introduction

Problems of transport are of major concern to many organizations. Examples can be found in distribution centers transporting merchandize to their customers, in institutes which employ a large number of workers, taking care for their transportation, etc. Many variations of transport problems are studied in real life and in the academic literature. These are commonly NP-Hard. Heuristic and Metaheuristic methods were developed for solving such problems [2,3,9,11,14].

Here we present a unique transport problem we got familiar with at our college. The college has two campuses located at two southern Israeli cities, "Ashdod" and "Beer Sheva", denoted here by "A" and "B" respectively. The college logistics should manage

Elad Shufan
Physics Department, SCE, Shamoon College of Engineering, Beer-Sheva, Israel
E-mail: elads@sce.ac.il

Hagai Ilani
Department of Industrial Engineering and Management, SCE, Shamoon College of Engineering, Ashdod, Israel
E-mail: hagai@sce.ac.il

Tal Grinshpoun
Department of Software Engineering, SCE, Shamoon College of Engineering, Beer-Sheva, Israel
E-mail: talgr1@sce.ac.il

the transportation of lecturers living in city "A" to campus "B" and vice-versa. Few weeks before the semester starts, each lecturer, knowing her teaching hours, requests for the time she wants to leave her hometown (either "A" or "B") and the time she wants to go back home from the distanced campus ("B" or "A"). The college has a finite number of vehicles of different capacities (usually between 5-9 passengers), and an optimal set of transports is needed to be found. Optimal here refers to the waiting time of the lecturers. Although motivated by a real life problem, the focus of the paper is theoretical.

The two-campus transport problem (TCTP) can be viewed as a version of the well known dial-a-ride problem (DARP) [6,9]. In a DARP instance, a set of ride-requests is given, each ride has a source and a destination. The rides should be operated by a fleet of vehicles in order to optimize some objective function. The complexity of different versions of DARP is discussed by Paepe et. al. [10]. TCTP is a special case of a DARP, where all the sources and destinations are either "A" or "B". We also distinguish between the two different types of ride-requests (or passengers), which we denote "s-type" and "r-type". "S-type" passengers are being transported to teach, i.e., they are just **starting** their daily teaching obligation, while passengers of "r-type" have already finished their lecture and are **returning** back home. Some DARPs also distinguish between "inbound" and "outbound" requests, correspond to "r" and "s" types respectively. The distinction is expressed by a time-window which accompanies the request. A lecturer who has finished teaching at t_r , determines the ride's *release* time, and a largest possible time window is $[t_r, t_{fin}]$, where t_{fin} is the last possible time for the transport departure, determined by the end of the day. A lecturer who is going to teach determines a *deadline* time t_d , and the corresponding largest time window is $[t_{initial}, t_d]$, where $t_{initial}$ is the beginning-of-the-day time. The most relevant objective function in the DARP literature is the latency, determined by the sum of the completion time of all the rides (denoted $\sum C_i$ in [10]). As for this latency function, "the sooner the better", the request should be fulfilled as early as possible. However, the TCTP is different because when an "s-type" request is considered, the lecturer usually wants the request to be executed as close as possible to the deadline, definitely not as soon as possible. The TCTP objective we take treats the dissatisfaction of customers with regard to the aberration between the real pick-up time and the requested time, distinguishing between s-type and r-type requests. This is usually not considered in a DARP. The TCTP is therefore a special case of a DARP which is not considered in [10]. Besides a simple one directional case of TCTP (section 3), which is indeed simple, the complexity of the TCTP is yet not known.

Most of the article treats the one-directional case, namely the arrangement of N passengers into k groups, each group represents a transport going from one city to the other, without taking into account the fact that the same vehicles are needed to take passengers also on the other direction. In other words, we do not check the availability of the transport. In what follows, when discussing the one-directional case, we treat the transportation of N passengers from "A" to "B", assuming that k vehicles all wait at "A" for their departure.

We state three main levels of problem complexity, with increasing difficulty:

1. The one-directional case, with passengers of one type, i.e, either all the passengers are going to teach, or all passengers have just finished lecturing and want to go back to their hometown. We mathematically define this relatively simple problem

and, for a fleet of uniform capacity, suggest a complete polynomial time solution. The solution is based on a reduction of an ordered set optimal partition problem [4] to the known shortest path problem [1, 8].

2. The one-directional case, with passengers of two types. We present the problem mathematically and suggest an algorithm for obtaining the complete space solution. This is done by generation of the linear extensions of a partially ordered set and a reduction to the shortest path problem. This provides a practical solution to a relatively small solution space and uniform capacity fleet.
3. The general case of two directions, with two types of passengers, and a fixed number of vehicles of known capacities.

Note that each of the three sub-problems can have some variations – passengers might have a different level of importance, the two-directional problem can have only one passenger type, etc.

After giving some basic notations and principles on section 2, we describe the simple one-directional case (section 3) and the complex one (section 4). On the final section we conclude.

2 Definitions and problem statement

Two sets are given $S = \{s_1, s_2, \dots, s_m\}$ and $R = \{r_{m+1}, r_{m+2}, \dots, r_{m+n}\}$, representing the requested departure times of s-type and r-type lecturers, respectively. The target is to divide $S \cup R$ into k subsets, each represents one transport, and to define the departure time of each transport, so as to minimize the wasted time lectures wait for their transport. A wasted time of one lecturer is defined as the difference between the requested time and the actual departure time of the transport the lecturer was placed at. Note that we actually divide the set $\{1, 2, \dots, N\}$, with $N = m + n$.

2.1 Objective function

The college logistics are expected to minimize the number of transports, as well as the wasted time. These two objectives are in conflict – if we increase the number of transports, we decrease the wasted time (obviously when the number of transports is equal to the number of lecturers, no one waits, but the cost is high). Our aim is to find the best transport arrangement, for a given number of transports, k . The transport manager should start with the minimal k , compute the best arrangement, then increase k , check the improvement and decide if it justifies adding a transport. Iteratively, she goes on increasing k , until an optimal balance between time and money is achieved.

We consider two common objectives of minimizing the wasted time. The first, the additive approach, is to minimize the *sum* of wasted times of all the lecturers. The second, the min-max approach, is to minimize the *maximal* wasted time. Let $\{i_1, i_2, \dots, i_a, i_{a+1}, i_{a+2}, \dots, i_b\}$, with $b \geq a$, be a subset representing the requested times $r_{i_1} \leq r_{i_2} \leq \dots \leq r_{i_a} \leq s_{i_{a+1}} \leq s_{i_{a+2}} \leq \dots \leq s_{i_b}$, of lecturers sharing the same transport. Inequality $r_{i_a} \leq s_{i_{a+1}}$ is a necessary condition – an r-type lecturer, who wants to return after finishing her lecture, cannot share a transport with an s-type lecturer who needs to depart earlier. Denote the departure time by t . The contribution of the given transport to the additive objective function $f_{Additive}$ is given by

$$\sum_{j=1}^a (t - r_{i_j}) + \sum_{j=a+1}^b (s_{i_j} - t)$$

Considering the min-max objective function, f_{MinMax} , the maximal wasted time of this transport is

$$\max\{(t - r_{i_1}), (s_{i_b} - t)\}$$

The *optimal* departure time of the given transport t_{OPT} is determined in the following way: if $b - a = 0$ (i.e. no starting lecturers), then $t_{OPT} = r_{i_a}$. If $a = 0$ (no returning lecturers), $t_{OPT} = s_{i_{a+1}}$. Otherwise ($0 < a < b$), t_{OPT} depends on the objective function. When the **additive** objective function is considered

$$t_{OPT} = \begin{cases} r_{i_a} & a > b - a \\ (r_{i_a} + s_{i_{a+1}})/2 & a = b - a \\ s_{i_{a+1}} & a < b - a \end{cases} \quad (1)$$

This means that the majority rules: if there are more returning rather than starting lecturers, then as soon as the last r-type lecturer has done her duty, the transport leaves; if there are more s-type lecturers, the transport departs at the requested time of the earliest starting lecturer. As long as the **min-max** objective function is considered, the optimal time is given by

$$t_{OPT} = \begin{cases} \frac{r_{i_1} + s_{i_b}}{2} & r_{i_a} \leq \frac{r_{i_1} + s_{i_b}}{2} \leq s_{i_{a+1}} \\ r_{i_a} & \frac{r_{i_1} + s_{i_b}}{2} < r_{i_a} \\ s_{i_{a+1}} & \frac{r_{i_1} + s_{i_b}}{2} > s_{i_{a+1}} \end{cases} \quad (2)$$

The ideal departure time, minimizing $\max\{(t - r_{i_1}), (s_{i_b} - t)\}$, is the average of earliest and latest requested times. However, the departure time has to satisfy $r_{i_a} \leq t \leq s_{i_{a+1}}$ since the transport cannot leave before the last returner has finished teaching (r_{i_a}), or after the earliest time of a starting lecturer (otherwise she will be late). If the average is larger than r_{i_a} or smaller than $s_{i_{a+1}}$, the optimal time for departure will be r_{i_a} or $s_{i_{a+1}}$ respectively.

Note that for both objective functions the optimal departure time of a given one-directional transport is completely determined. We are then left only with searching for the optimal *partition*.

2.2 Partitioning

The notations and definitions of finding an optimal partition follow Chang et al. [5]. Given a linear ordered set $\Theta = \{\theta_1 < \theta_2 < \dots < \theta_N\}$, a *partition* is a collection of pairwise disjoint subsets $\Pi = (\pi_1, \pi_2, \dots, \pi_k)$ whose union is $\{1, 2, \dots, N\}$. The partition *size* is $|\Pi| = k$. Let $n_i = |\pi_i|$, the sequence $(n_1, n_2, \dots, n_k)$ is referred to as the partition *shape*. A *consecutive* partition is one whose all partition *parts*, π_i , consists of consecutive integers. Let f be a function defined on all possible partitions. The

optimal partition problem is that of finding a partition which minimizes or maximizes f .

When considering the TCTP, we search for a partition of the set of times $S \cup R$. The partition size k represents the number of transports. Each partition part represents one transport, with partition size being equal to the number of passengers in the relevant transport. If the j th transport is implemented using a vehicle of capacity C_j , the j th partition size must evidently satisfy $n_j \leq C_j$.

2.3 Definition of a solution

A solution for the one-directional problem is a pair (Π, T) consists of a partition of $S \cup R$, $\Pi = (\pi_1, \pi_2, \dots, \pi_k)$, and a sequence of departure times $T = (t_1, t_2, \dots, t_k)$, with t_i denoting the departure time of transport π_i for all $i = 1, 2, \dots, k$. We assume, without loss of generality, that for a given solution $t_1 \leq t_2 \leq \dots \leq t_k$.

In section 2.1 we have shown how to define the departure times once the partition is given. Hence, for the one-directional case, Π determines T . This will no longer be true for the two-directional case, because at time t_{OPT} the car may not be available. In section 4 we shall present an algorithm for finding all the linear orders of $S \cup R$, so that the optimal partition is a consecutive one. If only consecutive partitions of an ordered set Θ are considered, the partition shape completely describes the partition. Given a linear ordered set Θ and a partition shape $(n_1, \dots, n_k)$, the transport arrangement is set, with n_1 passengers on the first transport, n_2 on the second, etc. We still need to find the optimal order-and-shape for a given objective function.

3 The simple one-directional case

The simple case is one with either $R = \emptyset$ or $S = \emptyset$ or $s_n < r_{n+1}$. In the latter scenario the lecturers' wish-list divides into two separate lists, the s-type and the r-type lists (a mixed transport cannot be applied). For all three scenarios, given a time-list of size N , one should first arrange the list in increasing order. If lecturer i have asked to depart earlier than lecturer j , it does not make sense to arrange the j th lecturer before the i th one (all lecturers have equal priorities). The optimal solution is then given by a consecutive partition. For k transports and vehicles of a same capacity C , the optimal partition problem can be solved by a reduction. Chakravarty et al. [4] have shown that if f is a minimum additive objective function, whose optimal partition is known to be consecutive, the optimal partition is then found by a reduction to the shortest path problem [1,8]. A directed graph with vertices $\{1, 2, \dots, N, N+1\}$ and arcs (i, j) , with $i < j$, is built. A weight $f(\{i, i+1, \dots, j-1\})$ is assigned for each arc (i, j) . A directed path from 1 to $(N+1)$ represents a possible consecutive partition. The capacity limit C is expressed by including the arc only if $j - i \leq C$. Since the maximal partition size is restricted by k , the number of transports, the path should not consist of more than k arcs. One should then use the Bellman-Ford algorithm [1] and stop the procedure after k iterations. A simple exercise in Bellman-Ford's algorithm shows that the reduction can also be applied for the min-max objective function.

Consider an example with $N = 7$ s-type lecturers. In this example the time-list is given by $S = \{7, 9, 10, 11, 13, 15, 15\}$ ($R = \emptyset$), the number of transports is $k = 3$, and each transport has a maximal capacity of $C = 3$ passengers. The reduction to the

shortest path problem is illustrated in Figure 1. Each vertex represents a single lecturer in S , except for vertex 8, which represents $N + 1$. The other vertices are numbered according to their order in S . Additionally, each vertex is placed along a time-line to demonstrate the values in S .

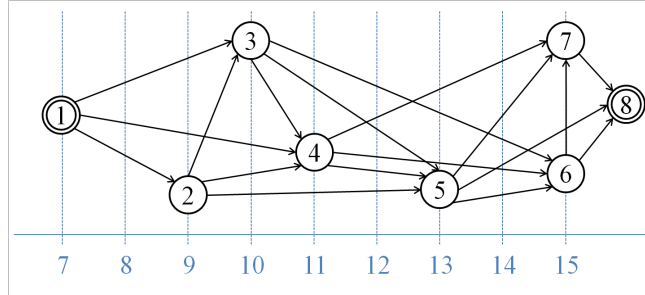
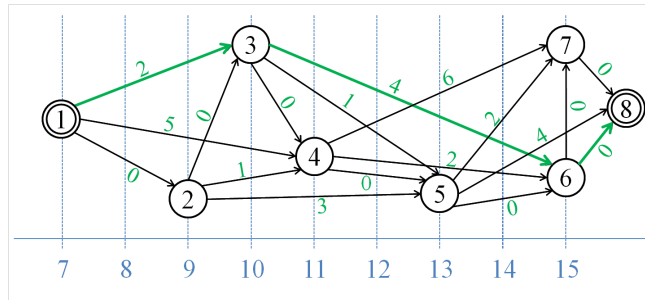
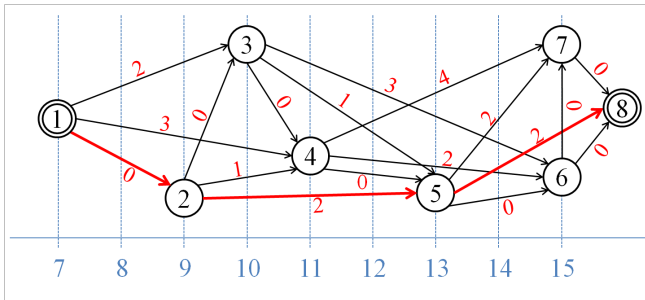


Fig. 1 The reduction to the shortest path problem for a simple one-directional example.

The weights of the arcs are given according to the chosen objective function. Figure 2 depicts the resulting weighted graphs when considering the objective functions $f_{Additive}$ and f_{MinMax} . Note that the resulting shortest path (i.e., optimal partition) is different for each objective function.



(a)



(b)

Fig. 2 The weighted graph and resulting shortest path for a simple one-directional example with the objective functions: (a) $f_{Additive}$; (b) f_{MinMax} .

The depicted reduction is applicable whenever the capacity C is the same for all vehicles. A problem arises when vehicles of different capacities, $\{C_j\}$, are considered, since the number of outgoing edges from each vertex in the reduction's graph is no longer unambiguous. One may attempt solving this problem by adding edges with respect to the maximal possible capacity, i.e., drawing the graph for a uniform capacity scenario with $C = \max(\{C_j\})$. The other smaller capacities are reflected in the reduction by imposing additional constraints, during the search for the shortest path, which are related to the vehicles that have already been used. However, this solution also fails, since Bellman-Ford, as well as other shortest path algorithms, relies on the *optimal-substructures property* [7]. According to the property, subpaths of shortest paths are also shortest paths. This clearly does not hold when the aforementioned constraints are imposed. Nevertheless, when the capacities' variance is small, solving a uniform problem with $C = \max(\{C_j\})$ might practically work. For a large enough k , many of the possible arrangements will still be valid, hopefully including the optimal ones.

Another inapplicable scenario is one in which the passengers are prioritized. Adding priorities inherently contradicts the consecutiveness of the optimal partition, which is an essential precondition of the simple one-directional case. Nevertheless, one may think of a weakened notion of prioritization, in which the priorities do not change the order of the passengers. An organization may decide, for the sake of appearance, not to allow a senior lecturer to take the place in a transport of a junior lecturer that finished work before her, but at the same time find an arrangement in which senior lecturers generally wait less time than others. In this case the priorities only affect the weights of the edges, so the reduction clearly holds.

4 The complex one-directional case

The case with $R, S \neq \emptyset$ and $s_n \geq r_{n+1}$ (i.e., the last s-type lecturer does not depart earlier than the first r-type lecturer) is rather complex. Unlike the previously discussed simple case, the optimal partition, for the case of uniform capacity C , is not necessarily obtained by a consecutive partition of the *increasingly* ordered set $S \cup R$. We demonstrate this with an example.

Consider $S = \{8, 13, 14, 19\}$ and $R = \{16, 21, 21\}$, with $k = 3$ transports and vehicle capacity of $C = 3$ passengers. Arranging $S \cup R$ with increasing order stems $\Theta_0 = \{\overrightarrow{8}, \overrightarrow{13}, \overrightarrow{14}, \overrightarrow{16}, \overrightarrow{19}, \overrightarrow{21}, \overrightarrow{21}\}$. Rightward and leftward arrows denote s-type and r-type lecturers respectively. The only possible consecutive partition is $(\pi_1, \pi_2, \pi_3) = ((1, 2, 3), (4, 5), (6, 7))$. This partition describes the transports $(\overrightarrow{8}, \overrightarrow{13}, \overrightarrow{14})$, $(\overrightarrow{16}, \overrightarrow{19})$, and $(\overrightarrow{21}, \overrightarrow{21})$, with objective functions $f_{Additive} = 14$ and $f_{MinMax} = 6$. However, better partitions exist. The additive optimal partition is $\{\{1\}, \{2, 3, 5\}, \{4, 6, 7\}\}$, describing the transports $(\overrightarrow{8})$, $(\overrightarrow{13}, \overrightarrow{14}, \overrightarrow{19})$, and $(\overrightarrow{16}, \overrightarrow{21}, \overrightarrow{21})$, with $f_{Additive} = 12$. The min-max optimal partition has $f_{MinMax} = 5$, with the partition given by $\{\{1, 2\}, \{3, 5\}, \{4, 6, 7\}\}$, describing the transports $(\overrightarrow{8}, \overrightarrow{13})$, $(\overrightarrow{14}, \overrightarrow{19})$, and $(\overrightarrow{16}, \overrightarrow{21}, \overrightarrow{21})$.

Both optimums are non-consecutive with respect to the set Θ_0 , but do form a consecutive partition with respect to $\Theta_1 = (\overrightarrow{8}, \overrightarrow{13}, \overrightarrow{14}, \overrightarrow{19}, \overrightarrow{16}, \overrightarrow{21}, \overrightarrow{21})$. Figure 3 displays the reduction and resulting shortest path for the additive objective function, $f_{Additive}$. The vertices in the figure are numbered according to Θ_1 . Note that some vertices have outdegree or indegree smaller than C , due to the s-r restriction.

In the spirit of the above example we suggest the following procedure for the complex case. The set $S \cup R$ should first be arranged in a manner in which the optimal

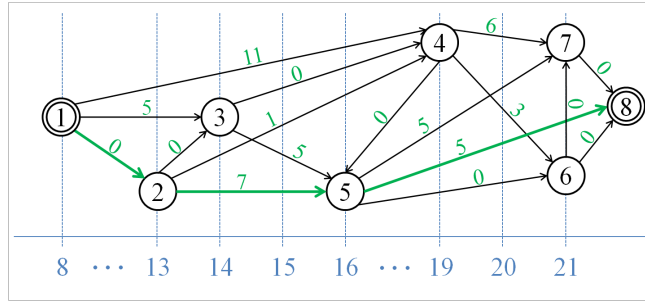


Fig. 3 The weighted graph and resulting shortest path for a complex one-directional example with the additive objective function.

solution is achieved by a consecutive partition. As discussed in section 3, when the optimal partition is indeed consecutive, it can be found using a polynomial-time reduction.

4.1 An algorithm for solving the complex case

We define an order relation, $\prec$, over the set $S \cup R$ by

- for $s_i, s_j \in S$, $s_i < s_j \rightarrow s_i \prec s_j$
- for $r_i, r_j \in R$, $r_i < r_j \rightarrow r_i \prec r_j$
- for $s_i \in S$ and $r_j \in R$, $s_i < r_j \rightarrow s_i \prec r_j$

According to the first condition, if an s-type lecturer should depart before another s-type lecturer, she will appear prior on the list. The second condition implies the same for r-type lecturers. The last condition says that if an r-type lecturer wants to depart after an s-type lecturer, then the r-type will also appear after the s-type on the list (note that the two cannot share a transport). As oppose, even if an r-type lecturer finishes teaching before the requested departure time of an s-type lecturer, it is still possible for her to depart after the starting lecturer. It is always possible, in principle, to send all the returning lecturers at the last transport (though it is usually not an optimal arrangement).

Definition 1 An order of $S \cup R$ which is a linear extension of the order $\prec$ is called a *proper order*.

According to the following theorem, the proper orders and their consecutive partitions completely describe the problem, in the sense that the best consecutive partition of one of the proper orders, is the optimal partition.

Theorem 1 Given a set of times $S \cup R$, there exist a proper linear order, $\tilde{\prec}$, and a partition Π which is optimal (with regards to any of the two mentioned objective functions), so that Π is a consecutive partition with respect to $\tilde{\prec}$.

The proof of theorem 1 is given in section 4.2.

We now present an algorithm for solving a uniform capacity one-directional complex case:

1. Given a time list $S \cup R$, create all its proper linear extensions. Algorithms for generating the linear extensions of a partial order exist in the literature [12,13].
2. For each proper order, build the reduction graph by the method described in section 3, keeping in mind that some arcs are impossible due to the s-r restrictions (in a specific transport all the s-type requested times should not be later than the r-type times).
3. Find the shortest path in the graph, by a Bellman-Ford algorithm, restricted to k steps. That shortest path describes the best k transports arrangement for the given order.

The algorithm efficiency depends on several factors: (i) the number of linear extensions of the given order; (ii) the efficiency of linear extensions generation; and (iii) the efficiency of finding the best solution described by the shortest path.

As for the last issue, a Bellman-Ford algorithm finds the optimal path in polynomial time $O(NkC)$. The generation of linear extensions is also efficiently done by the known algorithms, in time which is linear in the number of extensions. The efficiency is disturbed only by the number of linear extensions. As close as the given case is to the simple one, the number of extensions will be relatively small. The whole algorithm is then efficient. On the opposing side lies the worst case scenario – when the last returner should depart before the first starting lecturer. When this occurs, every linear order, which is S-ordered and R-ordered, is a proper linear extension. For an even N , the number of extensions is maximal if $n = m = N/2$ (the number of r-type equals the number of s-type lecturers). Then, the number of extensions is $\binom{2n}{n}$, which is exponential in n . The algorithm will then be restricted to n of limited size. In addition, the algorithm is wasteful in the sense that some solutions might be generated more than once. The extreme case is when $k = N$. The optimal solution for this case is to send a lecturer per transport. This solution will be repeated in all of the generated linear extensions.

4.2 Definitions extension and proof of theorem 1

We summarize and extend the previously given notations and definitions. The subset collection $\Pi = (\pi_1, \pi_2, \dots, \pi_k)$, denotes a partition of size k , with $\bigcup_{i=1}^k \pi_i = \{1, 2, \dots, N\}$. To keep the notations simpler, we shall identify Π with a partition of the actual requested times. We then define $S_i = S \cap \pi_i = \{s_{i1}, s_{i2}, \dots, s_{i\sigma_i}\}$, where $\sigma_i = |S_i|$ is the number of s-type passengers in the i-th transport, and, by definition, $s_{i1} \leq s_{i2} \leq \dots \leq s_{i\sigma_i}$. Similarly, for the r-type passengers, we have $R_i = R \cap \pi_i = \{r_{i1}, r_{i2}, \dots, r_{i\rho_i}\}$, $\rho_i = |R_i|$, and $r_{i1} \leq r_{i2} \leq \dots \leq r_{i\rho_i}$.

Definition 2 (i) If $r_{i\rho_i} \leq s_{i1}$ for every partition part π_i (where $i = 1, 2, \dots, k$) the partition is said to be **feasible**.

The feasibility of a partition guarantees that the last returning lecturer does not finish after the requested time of the first starting lecturer.

(ii) A solution (Π, T) is called a **feasible solution** if $r_{i\rho_i} \leq t_i \leq s_{i1}$. If (Π, T) is a feasible solution, the partition is also feasible. In addition, the time of departure is possible.

We define the objective functions of section 2.1 explicitly. Given a solution (Π, T) , the additive objective function is

$$f_{Additive}(\Pi, T) = \sum_{i=1}^k f_A^{(i)}(\pi_i, t_i) \quad (3)$$

where

$$f_A^{(i)}(\pi_i, t_i) = \sum_{r \in \pi_i} (t_i - r) + \sum_{s \in \pi_i} (s - t_i) \quad (4)$$

The min-max objective function is similarly defined by

$$f_{MinMax}(\Pi, T) = \max_{i=1}^k \{f_M^{(i)}(\pi_i, t_i)\} \quad (5)$$

where $f_M^{(i)}(\pi_i, t_i)$ is the largest wasted time in the i -th transport, and is given by

$$f_M(\pi_i, t_i) = \max\{\max_{s \in S_i} \{s - t_i\}, \max_{r \in R_i} \{t_i - r\}\} \quad (6)$$

We can always take the set T to be the optimal set, as given in equations (1) and (2). Nevertheless, for the sake of generality and proof simplicity, we present the definitions for a general possible sequence of departure times T .

Definition 3 A feasible solution (Π, T) defines the following ordered set

$$(r_{11}, r_{12}, \dots, r_{1\rho_1}, s_{11}, s_{12}, \dots, s_{1\sigma_1}, r_{21}, r_{22}, \dots, r_{2\rho_2}, s_{21}, s_{22}, \dots, s_{2\sigma_2}, \dots, \\ \dots, r_{k1}, r_{k2}, \dots, r_{k\rho_k}, s_{k1}, s_{k2}, \dots, s_{k\sigma_k})$$

This order is called the **canonical order** of $S \cup R$ defined by (Π, T) .

Definition 4 Given a partition Π and two requested times, s and s' , we define an (s, s') -**swap**, as the action of exchanging s and s' in the corresponding parts of Π . The partition obtained by an (s, s') -swap, is denoted by $\Pi^{s \leftrightarrow s'}$. Explicitly, if $s \in S_i$ and $s' \in S_j$, we define the parts of $\Pi^{s \leftrightarrow s'} = \{\pi_1^{s \leftrightarrow s'}, \dots, \pi_k^{s \leftrightarrow s'}\}$ by

$$\pi_l^{s \leftrightarrow s'} = \begin{cases} (\pi_i \setminus \{s\}) \cup \{s'\} & \text{if } l = i \\ (\pi_j \setminus \{s'\}) \cup \{s\} & \text{if } l = j \\ \pi_j & \text{else} \end{cases} \quad (7)$$

Let Π be a feasible partition. It does not follow that $\Pi^{s \leftrightarrow s'}$ is also feasible. A swap is called a **feasible swap**, if $\Pi^{s \leftrightarrow s'}$ is a feasible partition. We similarly define $\Pi^{r \leftrightarrow r'}$ for two given returning times r and r' .

Lemma 1 Let (Π, T) be a feasible solution. Let s and s' be two starting times such that $s \in S_i$, $s' \in S_j$, $i < j$ and $s \geq s'$. Then

1. $(\Pi^{s \leftrightarrow s'}, T)$ is also a feasible solution.
2. $f_{MinMax}(\Pi^{s \leftrightarrow s'}, T) \leq f_{MinMax}(\Pi, T)$.
3. $f_{Additive}(\Pi^{s \leftrightarrow s'}, T) = f_{Additive}(\Pi, T)$.

- Proof* 1. The inequalities for the returning times are immediate because (Π, T) is feasible. Hence, we should only show that $t_i \leq s'$ and $t_j \leq s$. The first inequality is valid since $i < j \Rightarrow t_i \leq t_j$ and (Π, T) is feasible, therefore $t_j \leq s'$. This latter inequality together with $s \leq s'$ leads to $t_j \leq s$.
2. Because of the swap, the waited times of the s and s' lecturers change from $s - t_i$ and $s' - t_j$ to $s - t_j$ and $s' - t_i$, respectively. Anyway, the maximal time out of the four possibilities is $s - t_i$. Therefore the maximal waited time has not been increased as a result of the swap.
3. $f_{Additive}(\Pi^{s \leftrightarrow s'}, T) = f_{Additive}(\Pi, T) - (s - t_i) + (s - t_j) - (s' - t_j) + (s' - t_i) = f_{Additive}(\Pi, T)$.

Definition 5 A partition Π is said to be **S-ordered** if for any pair $\pi_i, \pi_j \in \Pi$, $i < j \Rightarrow s \leq s'$, where $s \in S_i$ and $s' \in S_j$. We similarly define an **R-ordered** partition.

Corollary 1 An optimal partition which is S-ordered and R-ordered, always exists.

Proof Let (Π, T) be an optimal partition. If the partition is not S-ordered then there exist $\pi_i, \pi_j \in \Pi$, with $i < j$, and two elements $s \in S_i$ and $s' \in S_j$, such that $s \geq s'$. According to Lemma 1 the solution $(\Pi^{s \leftrightarrow s'}, T)$ is feasible and optimal for the two discussed objective functions. If that new partition is still not S-ordered, we continue swapping until an S-ordered partition is obtained. Similarly we lead the partition to its R-ordered form.

We are now ready to give the proof for theorem 1.

Proof (Theorem 1) Let us take the canonical order defined by an optimal solution (Π, T) , with an S-ordered and R-ordered partition (according to corollary 1 such solution exists). The partition is obviously consecutive with shape $(\rho_1 + \sigma_1, \rho_2 + \sigma_2, \dots, \rho_k + \sigma_k)$. We prove that this canonical order is a proper order, i.e., if we take $\pi_i, \pi_j \in \Pi$, then any two elements $q \in S_i$ or R_i , $q' \in S_j$ or R_j , are ordered with respect to $\prec$. The partition is S-ordered and R-ordered, therefore we only have to check the third condition of $\prec$. We take two elements $s \in S_i$ and $r \in R_j$ such that $s < r$. We should prove that $i < j$. Since (Π, T) is a feasible solution $i \neq j$. We assume indirectly that $i > j$. Then, $s \geq t_i \geq t_j \geq r$ in contradiction to $s < r$.

5 Discussion

We have mathematically modeled the TCTP, and gave an algorithm for solving the one-directional uniform case. The presented problem is a special variation of a DARP. In a general DARP the number of destinations is usually larger than two. Then, besides the need to arrange the partitions and the departure times, a major concern is the arrangement of routes which minimize the cost (or ride times). Our attitude is to find the best arrangement for a *fixed* number of transports, k , and the cost is evidently in proportion to k . We define a time-window, by treating "s-type" and "r-type" requests differently, but rather than taking the time window parameter (either deadline or release times) as a constraint, we inject it to the objective function, to reflect the dissatisfaction of the costumers. The common objective function that DARP researchers usually take refers to the total customer ride times. The fact that the TCTP has only one (forward and backward) route, enables us to treat the dissatisfaction more gently.

The models and methods in use for DARP problems reflect the complexity of the problem. It is common to use local search algorithms. For the TCTP, if the size of solution space increases considerably, local search might be relevant as well. Our presented work provides a natural definition of a neighborhood for a search algorithm. Two steps should be considered – the first is the swap as defined in equation 7. Given a solution (Π, T) one may consider the swaped partitions $(\Pi^{s \leftrightarrow s'}, T)$, $(\Pi^{r \leftrightarrow r'}, T)$, and also $(\Pi^{r \leftrightarrow s}, T)$. Another evident step is to change the partition shape, i.e., move a passenger from one transport to an adjacent one. When the two directional case is taken into account, a third step, that of changing the time of departure, might be considered as well, because that time is no longer fixed.

Our presented approach still needs to be extended. First, the main problem remained unsolved: the two-directional case, where car availability must be taken into account. Departure times are no longer explicit, because the car may be unavailable at the time which is optimal with regards to the objective function. Even the one-directional case itself can be made more complex, as we briefly discussed in section 3 by considering classes (priorities) or heterogenous capacities. How hard these problems really are? Obviously the simple one-directional case with uniform capacities is easy, but the question remains open for the other scenarios.

References

1. Bellman, R.E., On a routing problem, *Quarterly of Applied Mathematics* **16**, 87-90 (1958).
2. Bräysy, O., & Gendreau, M., Vehicle routing problem with time windows, Parts I and II, *Transportation Science* **39**, 104-139 (2005).
3. Bülbül, K., Ulusoy, G., & Sen, A., Classic transportation problems. In : Taylor D.G., (ed.), *Logistics engineering handbook*, CRC Press, Boca Raton, FL, USA, (2008).
4. Chakravarty, A.K., Orlin, J.B., & Rothblum, U.G., A partitioning problem with additive objective with an application to optimal inventory grouping for joint replenishment, *Operating Research* **30**, 1018-1022 (1982).
5. Chang, H., Hwang, F.K., & Rothblum, U.G., A new approach to solve open-partition problems, accepted to the *Journal of Combinatorial Optimization*, available online.
6. Cordeau, J.F., & Laporte, G., The dial-a-ride problem: models and algorithms, *Ann. Oper. Res.* **153**, 2946 (2007).
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., & Stein, C., *Introduction to algorithms*, 3rd ed., MIT Press, Cambridge, MA (2009).
8. Dijkstra, E.W., A note on two problems in connection with graphs, *Numerische Mathematik* **1**, 269-271 (1959).
9. Golden, B., Raghavan, S., & Wasil, E., *The vehicle routing problem: latest advances and new challenges*, Springer, Boston, MA, (2008).
10. de Paepe, W.E., Lenstra, J.K., Sgall, J., Sitters, R.A., & Stougie, L., Computer-Aided Complexity Classification of Dial-a-Ride Problems, **I**, N (F)ORMS JOURNAL ON COMPUTING **16**(2)120-1322004.
11. Parragh, S.N., Doerner, K.F., & Hartl, R.F., A survey on pickup and delivery problems, Part I, *Journal für Betriebswirtschaft* **58**(1), 21-51 (2008); and Part II, *Journal für Betriebswirtschaft* **58**(2), 81-117 (2008).
12. Pruesse, G., & Ruskey, F., Generating linear extensions fast, *SIAM Journal on Computing* **23**, 373-386 (1994).
13. Varol, Y.L., & Rotem, D., An algorithm to generate all topological sorting arrangements, *The Computer Journal* **24**, 83-84 (1981).
14. <http://neo.lcc.uma.es/radi-aeb/WebVRP>

Hybridized Genetic Algorithms to Schedule Interfering Job Sets on a Single Machine

René Ramacher • Lars Mönch

Abstract In this paper, we study a bicriteria single machine scheduling problem. The jobs belong to two disjoint sets. We consider the total weighted completion time (TWC) as the performance measure for the jobs of the first set, whereas the maximum lateness (L_{max}) is taken as the objective for the second set. All jobs are ready at time zero. They have to be scheduled non-preemptively on the single machine. Our aim is to generate the set of all non-dominated solutions. Because of the NP-hardness of the problem, we propose heuristics based on the Non-Dominated Sorting Genetic Algorithm (NSGA)-II and random key encoding. We show by computational experiments on randomly generated problem instances that the quality of the algorithms can be improved significantly when problem specific properties are used.

1 Introduction

Starting by the pioneering work by Peha [13] and Agnetis et al. [1] scheduling problems with interfering job sets have attracted recently many researchers (cf. [4,8] amongst others). Scheduling interfering job sets refer to a situation where different classes of jobs compete for the same resources. Different job sets might represent different customers, called agents in the remainder of this paper. The difference among the agents is modeled by a different performance measure for each of the job sets. For example, production jobs and prototyping jobs have to share the same expensive bottleneck machines in semiconductor wafer fabrication facilities. While the production department is interested in minimizing cycle time related performance measures in this situation, the production engineering department intends to minimize the L_{max} value of its jobs.

This research is motivated by scheduling problems discussed by Baker and Smith [3]. They show that the single machine interfering job scheduling problem with total completion time (TC) and L_{max} criterion is in polynomial time solvable, while the corresponding problem with TWC and L_{max} criteria is NP-hard. A linear combination of the two criteria is considered assuming a priori information on the weights of the two criteria. In this paper, we propose efficient heuristics for the NP-hard problem. However, in contrast to [3], we are interested in determining the set of all non-dominated solutions. Therefore, we will use the NSGA-II algorithm together with a random key representation of the sequencing problem. We show by a large set of randomly generated problem instances that the proposed approach leads to high-quality solutions in reasonable computational time when problem specific properties are taken into account.

The paper is organized as follows. In Section 2, we describe the researched problem. Because the problem is NP-hard, we propose several heuristics based on GAs in Section 3. We show especially certain structural properties of solutions of the researched problem. Finally, we present the results of computational experiments in Section 4.

René Ramacher
University of Hagen
E-mail: Rene.Ramacher@fernuni-hagen.de

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

2 Problem Statement and Related Work

We discuss a scheduling problem for interfering jobs sets. We consider n jobs that are available for processing on a single machine. No preemption of a job is allowed after it starts processing. We denote the processing time of job j by p_j . Job j might have a due date d_j and also a weight w_j that is used to model the importance of the job. The completion time of job j is denoted by C_j . We consider two different agents A_1 and A_2 . The set of all n jobs consists of two disjoint classes $C1$ and $C2$. We have n_i jobs in Ci . The jobs of class Ci are assigned to agent A_i . The objective of A_1 is TWC of the jobs belonging to $C2$ in a schedule S :

$$TWC(S) := \sum_{j \in C1} w_j C_j. \quad (1)$$

We denote the lateness of job j by $L_j := C_j - d_j$. The corresponding objective of A_2 is L_{max} given by

$$L_{max}(S) := \max\{L_j \mid j \in C2\}. \quad (2)$$

Using the $\alpha \mid \beta \mid \gamma$ notation from scheduling theory the researched scheduling problem can be described as follows:

$$I \mid inter \mid ND(TWC, L_{max}), \quad (3)$$

where we denote by *inter* the interfering job sets. Furthermore, we indicate that we are interested in determining the set of non-dominated solutions by $ND(TWC, L_{max})$. Let $\tilde{S}$ the set of feasible solutions of Problem (3). We denote by $L_{max}(S)$ the objective value for the schedule $S \in \tilde{S}$ corresponding to $C2$ and by $TWC(S)$ the corresponding objective value that is related to $C1$. A schedule S^* is called non-dominated or Pareto optimal if there exists no other schedule $S \in \tilde{S}$ such that $L_{max}(S) \leq L_{max}(S^*)$ and $TWC(S) \leq TWC(S^*)$, and at least one of the two inequalities is strict.

It is proven in [3] that $I \mid inter \mid TWC + \theta L_{max}$, a special case of Problem (3), is NP-hard. A GA and Simulated Annealing are proposed in [14] for this problem. A polynomial time algorithm is presented for $I \mid inter \mid TC + \theta L_{max}$ in [15]. A heuristic for $I \mid inter \mid ND(TC, \sum U_j)$ is described in [12]. Here we denote by $\sum U_j$ the number of late jobs. GAs for $P \mid inter \mid ND(TC, C_{max})$ are proposed in [4], where we denote by C_{max} the makespan. Because the NP-hardness of Problem (3) we have to look for efficient heuristics. However, to the best of our knowledge, there are no such heuristics available for Problem (3) in the literature.

3 Heuristic Solution Approach

Three different dominance properties that exploit the structure of the problem are introduced in Subsection 3.1. In Subsection 3.2, we briefly describe a hybrid NSGA-II scheme. An NSGA-II approach that takes the properties into account is presented in Subsection 3.3.

3.1 Structural Properties

We start by introducing the notion of blocks that is important for the structural properties. A schedule S of interfering job sets can be obtained by interleaving sequences of jobs of the different Ci . Jobs of the same Ci that are sequenced in a consecutive manner form a block S^{Ci} .

The first property is related to reducing the TWC value of a given schedule by swapping appropriate jobs of $C1$ and of $C2$. It is based on the insight that for $j \in C2$ with $L_j < L_{max}$ the lateness of j can be increased to a certain amount without increasing the L_{max} value.

Property 1: Assume that the jobs $a \in C2$ and $b \in C1$ are adjacent and a is before b , such that

$$L_{max}(S) = C_a - d_a + \Delta \quad (4)$$

and

$$p_b \leq \Delta. \quad (5)$$

Then S can be changed into a schedule S' with $TWC(S') < TWC(S)$ and $L_{max}(S') = L_{max}(S)$ by exchanging a and b .

Proof: By exchanging a and b , a will be shifted to the right for $p_b \leq \Delta$ time units. Because of expression (4), the $L_{max}(S)$ value does not change. Since the completion times of the remaining jobs are not affected by swapping a and b , we obtain $L_{max}(S') = L_{max}(S)$. Due to the exchange of a and b , $C_b(S') = C_b(S) - p_a$ holds. Hence, we obtain by the same argument $TWC(S') < TWC(S)$. $\square$

Notice that we call $a \in C2$ delayable with respect to its adjacent job $b \in C1$ if it fulfills the two conditions (4) and (5). The second property deals with a situation where we swap only jobs within one block of $C1$ jobs. This result is basically a consequence of the optimality of the weighted shortest processing time (WSPT) rule for $I || TWC$.

Property 2: If there exists a block S_i^{C1} of $C1$ jobs in S , where the jobs are not sequenced in WSPT order, then S can be changed into a schedule S' with $TWC(S') < TWC(S)$ and $L_{max}(S') = L_{max}(S)$ by sorting the jobs in S_i^{C1} in WSPT order.

Proof: If the sequence of jobs in S_i^{C1} does not fulfill the WSPT order, then two consecutive jobs a and b exist with $w_a/p_a < w_b/p_b$ and a is processed before b . We obtain

$$w_a C_a(S') + w_b C_b(S') < w_a C_a(S) + w_b C_b(S) \quad (6)$$

by swapping the two jobs. Since the exchange of a and b does not affect the remaining jobs, the completion times of these jobs are not changed. $\square$

Finally, the next property pertains to every non-dominated point in the criteria space, where a point corresponds to a specific TWC and L_{max} value for jobs in $C1$ and $C2$ respectively.

Property 3: For every non-dominated point in the criteria space, there exists a schedule where all jobs belonging to $C2$ are sequenced in non-decreasing order of d_j , that is, in EDD order.

Proof: We assume that schedule S is non-dominated and that its $C2$ jobs are not in EDD order. Then there are two adjacent jobs $a, b \in C2$, such that a is sequenced before b and $d_a > d_b$. Because S is non-dominated, there is no block of $C1$ jobs between a and b . Let t the point of time where a starts in S . Then we obtain for the lateness $L_a(S) = t + p_a - d_a$ and $L_b(S) = t + p_a + p_b - d_b$. After swapping a and b , we have:

$$L_a(S') = t + p_a + p_b - d_a < L_b(S) \leq L_{max}(S), \quad (7)$$

$$L_b(S') = t + p_b - d_b < L_b(S) \leq L_{max}(S). \quad (8)$$

Therefore, because S is non-dominated, we obtain $L_{max}(S) = L_{max}(S')$ and $TWC(S) = TWC(S')$. $\square$

We will use the three properties to improve solutions obtained by metaheuristics.

3.2 NSGA-II Approach Hybridized with Generic Local Search

The NSGA-II scheme [6] is a well-established metaheuristic for multi-objective combinatorial optimization problems based on the principals of GAs. NSGA-II uses a non-dominated sorting procedure that determines the fitness of a solution s with respect to the number of solutions that dominate s . All solutions of a population P_t are partitioned into different sets using the non-dominated sorting procedure. Each single set is called a front. The front F_0 consists of all solutions that are not dominated by any other solution in P_t . In general, a front F_i contains only solutions that are not dominated by any solution of $P_t / \bigcup_{0 \leq j < i} F_j$. To construct the population P_{t+1} the solutions of P_t are selected starting from F_0 . Each solution of a front is selected until a certain front F_i does not completely fit into the next generation. The solutions of this front F_i are selected using the crowding operator to ensure a diversification of the solutions in the population. This is done by selecting solutions with respect to their distance to each other.

We use a permutation based representation of feasible schedules. A random key approach [5] is taken to encode the permutations of the n jobs of the combined job set. Each job is represented by an individual gene. A random number $z_j \sim U[0,1]$ is assigned to each gene. Sequences of jobs, i.e. feasible schedules, are derived by sorting the genes of a chromosome with respect to their keys in non-increasing order. A parameterized uniform crossover is used. Here, two parents are randomly selected from the entire population. Then a biased coin is tossed for each gene to determine which parent will contribute to the allele. We found that this crossover operator outperforms one-point and two-point crossover operators. We use a swap type mutation operator that exchanges two randomly selected genes in a chromosome.

The NSGA-II approach is hybridized with local search as proposed in [7]. The local search is intended to move solutions found by NSGA-II closer to the true Pareto front. To apply the local search procedure the objectives have to be combined into a single weighted objective function. In the case of the Problem (3), the combined objective function is:

$$F(S) = \frac{w_{TWC}^S}{w_{TWC}^S + w_{L_{max}}^S} TWC(S) + \frac{w_{L_{max}}^S}{w_{TWC}^S + w_{L_{max}}^S} L_{max}(S). \quad (9)$$

The two weights are chosen as follows:

$$w_{TWC}^S = \frac{TWC^{max} - TWC(S)}{TWC^{max} - TWC^{min}}, \quad (10)$$

$$w_{L_{max}}^S = \frac{L_{max}^{max} - L_{max}(S)}{L_{max}^{max} - L_{max}^{min}}, \quad (11)$$

where we denote the maximum and minimum value of an objective f by f^{max} and f^{min} respectively. It is assumed that $f^{min} \neq f^{max}$. The local search procedure is based on swapping of two jobs. A job at position k is exchanged with all jobs at position $k+1, \dots, \max(k+dist_{max}, n)$ in S . All jobs are considered. Transitive exchanges of jobs are possible. Moves are accepted only if they lead to an improvement of the combined objective (9). A single pass through the job set is repeated run_{max} times.

3.3 NSGA-II Approach Hybridized with Problem Specific Heuristics

In this section, we describe a heuristic to transform a schedule encoded by a chromosome of the NSGA-II scheme into a schedule which fulfills the properties shown in Subsection 3.1. The heuristic is suitable to be used as an alternative to the general purpose local search procedure within the hybrid NSGA-II approach. The heuristic applied to a single chromosome can be summarized as follows.

1. The $C2$ jobs are rearranged within the schedule encoded by a chromosome in EDD order. The rearrangement is carried out by exchanging consecutive $C2$ jobs starting by the first $C2$ job. $C1$ jobs are left on their positions as determined by the chromosome.
2. A delayable $C2$ job is identified and exchanged with its adjacent $C1$ job. Step 2 is repeated until no more delayable $C2$ jobs are to be left over.
3. The set of all $C1$ blocks is determined for the schedule S that results from Step 2. S is transformed into S' by rearranging the $C1$ jobs in WSPT order for each of the $C1$ blocks.

Note that Step 1 is due to Property 3, whereas Step 2 and Step 3 are justified by Property 1 and 2 respectively. The improvement of the TWC value obtained by Step 2 depends on the sequence of the $C1$ job determined by the schedule encoded within the chromosome. A delayable $C2$ job a is always swapped with its direct successor. It has to be noted that a swap with another of the following $C1$ jobs might lead to a larger TWC improvement. Therefore, the NSGA-II scheme is in charge to determine an appropriate sequence of the $C1$ jobs.

The three steps have to be performed in the order as we described them to achieve the best performance. Step 1 has to be executed prior to Step 2 because a schedule that does not fulfill Property 3 often lead to a large L_{max} value for its $C2$ jobs. Note that both Step 2 and Step 3 do not destroy the EDD order of the $C2$ jobs. Applying Step 2 to schedules with a large L_{max} value might lead to a situation where the $C2$ jobs are inadequately delayed. As a result, schedules with small L_{max} values are systematically excluded. Step 2 and Step 3 do not change the L_{max} value of the schedule. Moreover, Step 2 has to be followed by Step 3 because the performance of Step 2 is influenced to a large extent by the sequence of the $C1$ jobs obtained from the chromosome. Step 3 might rearrange the blocks of $C1$ jobs. Consequently, the sequence of these jobs encoded by a chromosome will be destroyed.

We continue with describing an initialization schema for the NSGA-II algorithm that makes sure that at least the two extremities of the Pareto-front are included within the initial population, i.e., the points that contain the smallest possible TWC value for $C1$ and the smallest possible L_{max} value for $C2$ jobs respectively. Due to the elitist selection mechanism of the NSGA-II scheme that guarantees that non-dominated solutions are included in subsequent generations, it is assured that these extremities are also included in the final generation and therefore it is guaranteed that they are also included within the resulting Pareto frontier.

We start by constructing a schedule that leads to a minimal TWC value. The schedule consists of a block of $C1$ jobs in the beginning, followed by a second block of $C2$ jobs. Applying Step 1 and Step 3 leads to a schedule with the smallest possible TWC value. Due to the EDD order of the jobs in the second block, the L_{max} value cannot be further reduced without increasing the TWC value which is itself minimal. Hence, the resulting schedule is Pareto optimal. To initialize a chromosome that encodes a schedule with minimal L_{max} value, all $C2$ jobs are used to form a block in the beginning of the schedule. The application of Step 1 ensures that the $C2$ jobs are sequenced in EDD order which makes sure that the L_{max} value is minimal. Furthermore, Step 2 and 3 are executed on the schedule to ensure that the schedule fulfills the Properties 1 and 2. Nevertheless, it is not guaranteed that the resulting schedule is Pareto optimal because of a possible improper interleaving of the jobs resulting from applying Step 2.

4 Computational Experiments

In our experiments, we are interested in comparing the performance of the heuristic approaches described in Section 3. Therefore, we start by presenting the used design of experiments in Subsection 4.1. Then we describe the implementation and the parameter setting of the heuristics in Subsection 4.2. Finally, the results of the computational experimentation are presented in Subsection 4.3.

4.1 Design of Experiments

We expect that the performance of the heuristics is influenced by the number of jobs in each job set, the tightness T , and the range R of the due dates of the $C2$ jobs. We generate a set of ten independent test instances for each factor combination. The used design of experiments is summarized in Table 1.

Table 1: Design of Experiments

Factor	Level	Count
n_1 / n_2	60/60, 100/60, 60/100, 100/100	4
Due Dates		
	$d_i \sim U[\mu(1 - R/2), \mu(1 + R/2)]$	
	$makespan = \sum_{j \in C1 \cup C2} p_j$	
	$\mu = makespan(1 - T)$	
Tightness	$T \in \{0.2, 0.8\}$	2
Range	$R \in \{0.2, 0.8\}$	2
Weight of jobs in $C1$	$w_j \sim DU[0.5, 1.5, 3, 8]$	1
Processing time	$p_j \sim DU[1, 3, 6, 9]$	1
	Total number of factor combinations	16
	Total number of problem instances	160

We hybridize NSGA-II with a local search approach and with the problem specific heuristics as described in Subsection 3.2 and 3.3. The resulting heuristics are called NSGA-II-LS and NSGA-II-H for abbreviation respectively. We perform three independent runs for each problem instance both with the NSGA-II-LS and the NSGA-II-H approach to obtain statistically reasonable results.

The obtained Pareto frontiers are denoted by $y_{NSGA-II-LS}$ and $y_{NSGA-II-H}$. Each Pareto front is generated out of all Pareto optimal solutions generated by the three independent runs for each approach. Furthermore, in order to assess the quality of the Pareto frontiers, we consider problem instances for $I | inter | ND(TC, L_{max})$. We use the exact algorithm proposed in [15] to solve the instances of $I | inter | TC + \theta L_{max}$. By taking θ from an equidistant grid with step size 0.01 over $[0.01, 1000]$, we generate a set of Pareto optimal solutions and use them as the Pareto front denoted as y_{opt} . Note again that each optimal solution for a problem instance of $I | inter | TC + \theta L_{max}$ is a Pareto optimal solution with respect to the objective TC, L_{max} .

Using the $\cup_{nondom}$ operator to indicate that dominated solutions are removed when performing the union operation, we derive the near to optimal Pareto front by

$$\gamma_{true} := \gamma_{NSGA-II-LS} \cup_{nondom} \gamma_{NSGA-II-H} \quad (12)$$

in the case of $I | inter | ND(TWC, L_{max})$ and by

$$\gamma_{true} := \gamma_{NSGA-II-LS} \cup_{nondom} \gamma_{NSGA-II-H} \cup_{nondom} \gamma_{opt} \quad (13)$$

for instances of $I | inter | ND(TC, L_{max})$. Following [11,16], we use cardinality as well as distance based measures for assessing each generated Pareto front. The cardinality based measure Overall Non-dominated Vector Generation (ONVG) determines how many solutions are in a Pareto front γ_{known} . We obtain:

$$ONVG := |\gamma_{known}|. \quad (14)$$

The measure OTNVG states how many solutions generated by an algorithm are Pareto optimal solutions with respect to γ_{true} :

$$OTNVG := |\{y \mid y \in \gamma_{known} \cap \gamma_{true}\}|. \quad (15)$$

The error ratio for solutions that are in γ_{known} but not in γ_{true} is defined by

$$Error := \sum_{i=1}^{ONVG} e_i / ONVG, \quad (16)$$

where $e_i := \begin{cases} 0 & \text{if } y_i \in \gamma_{known} \cap \gamma_{true} \\ 1 & \text{otherwise} \end{cases}$. An *Error* value close to zero means that a large number

of solution from γ_{known} is in γ_{true} . The distance based measure *dist* is used to determine the average distance of solutions in γ_{known} to the nearest solution in γ_{true} :

$$dist := \sqrt{\sum_{y \in \gamma_{known}} \left(\min_{y' \in \gamma_{true}} d(y, y') \right)^2} / ONVG, \quad (17)$$

where we measure the distance between two solutions y and y' by

$d^2(y, y') := \sum_{j=1}^2 \Theta_j^2 (f_j(y) - f_j(y'))^2$. Here, $\Theta_j = (f_j^{max} - f_j^{min})^{-1}$ is used for $j=1,2$. In case of $f_j^{max} = f_j^{min}$ we use $\Theta_j = 1$.

4.2 Implementation and Parameter Setting

The parameters of both heuristics are determined by trial and error based on extensive computational experiments. The probability of tossing a head of 0.5 is used for the parameterized uniform crossover. A mutation probability of 0.1 is selected in our experiments. Moreover, a population size of 180 and a total amount of 100 generations is applied within all experiments for NSGA-II type algorithms. We use $dist_{max} = 3$ and $run_{max} = 5$ in the local search procedure sketched in Subsection 3.2. Both NSGA-II-LS and NSGA-II-H are implemented in the C++ programming language using the MOMLib++ library [10]. All experiments are performed on a Pentium 4, 3.6 GHz PC with 2 GB RAM.

4.3 Results

The results obtained for Problem (3) are shown in Table 2. Instead of comparing all problem instances individually, the instances are grouped according to levels of factors like number of jobs and due date related factors. For example, results for $T = 0.2$ implies that all other factors have been varied, but the value of T has been kept constant at 0.2. Best results with respect to one of the two algorithms are marked in bold in all the result tables.

Considering the values for *Error* and *dist*, the NSGA-II-H approach outperforms the NSGA-II-LS approach clearly. The value for the *Error* measure points out that a large number of the solutions determined by NSGA-II-H are not dominated. Moreover, the value for *dist* identifies the front generated by NSGA-II-H approach as the front whose solutions have the closest distance to the solutions of the approximation of the true Pareto front. The value of the ONVG measure clearly shows that the NSGA-II-LS approach provides a significant smaller number of solutions to the generated Pareto front than the NSGA-II-H approach. The OTNVG values confirm the *Error* values by showing that a large number of solutions generated by NSGA-II-H belongs to γ_{true} . The obtained Pareto frontiers for a specific problem instance are shown for illustration purposes in Figure 1.

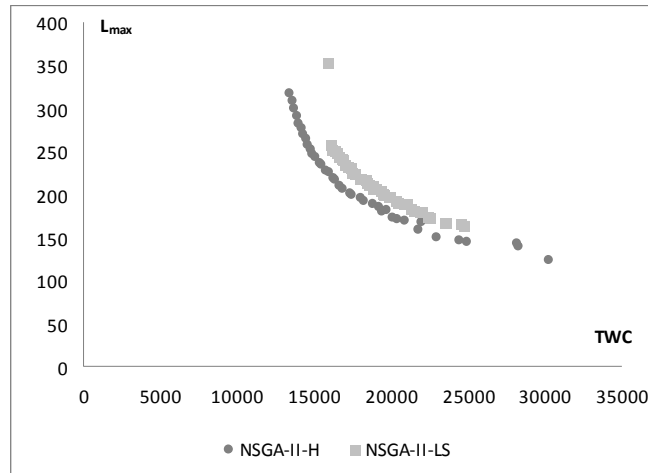


Figure 1: Pareto Frontiers for NSGA-II-H and NSGA-II-LS ($n_1 = n_2 = 60, R=T=0.8$)

The results obtained from the overall assessment are confirmed by analyzing the results obtained for the problem instances in the different groups. Considering the *Error* values with respect to T , the results show that the value for NSGA-II-LS decreases by 6 %, when T increases. In contrast, the *Error* value of NSGA-II-H increases by 3 % in the same situation, but it is still close to zero. Studying the effect of R , the obtained results show that the *Error* value of NSGA-II-LS increases by 7 % when R increases, while the error of NSGA-II-H decreases by 3 % in the same situation, but again it is almost zero.

Table 2: Computational Results for Problem (3)

Compare	Algorithm	ONVG	OTNVG	Error	dist	Time
$T=0.2$	NSGA-II-LS	50.362	4.125	0.877	0.025	171
	NGSA-II-H	99.475	99.425	<0.001	< 0.001	12
$T=0.8$	NSGA-II-LS	75.85	16.012	0.816	0.022	176
	NGSA-II-H	64.275	62.525	0.031	< 0.001	12
$R=0.2$	NSGA-II-LS	74.462	16.175	0.81	0.022	176
	NGSA-II-H	65.550	63.788	0.03	< 0.001	12
$R=0.8$	NSGA-II-LS	51.759	3.962	0.883	0.025	171
	NGSA-II-H	98.200	98.162	0.001	< 0.001	13
$n_1=60 / n_2 = 60$	NSGA-II-LS	61.700	9.025	0.863	0.018	97
	NGSA-II-H	91.300	90.700	0.010	< 0.001	11
$n_1=60 / n_2 = 100$	NSGA-II-LS	34.325	6.9500	0.796	0.014	171
	NGSA-II-H	73.700	73.050	0.012	< 0.001	12
$n_1=100 / n_2 = 60$	NSGA-II-LS	102.300	13.775	0.878	0.036	168
	NGSA-II-H	86.650	85.600	0.014	< 0.001	11
$n_1=100 / n_2 = 100$	NSGA-II-LS	54.100	10.525	0.850	0.026	258
	NGSA-II-H	75.850	74.550	0.027	< 0.001	15
Overall	NSGA-II-LS	63.106	10.069	0.847	0.024	173
	NGSA-II-H	81.875	80.975	0.016	< 0.001	12

The number of jobs does not have a large impact on the solution quality. In order to assess the proposed heuristics by comparing them with exact solutions, we present the results for $I | inter | ND(TC, L_{max})$, a special case of Problem (3), in the same manner in Table 3.

The results shown in Table 3 demonstrate that the Pareto front generated by NSGA-II-H leads to a smaller *Error* value than the Pareto front generated by NSGA-II-LS with respect to the true Pareto front. The *dist* values show that the solutions in $\gamma_{NSGA-II-LS}$ as well as the solutions in $\gamma_{NSGA-II-H}$ are close to the solutions of the true Pareto front. The differences between NSGA-II-LS and NSGA-II-H are smaller compared to the situation for Problem (3) presented in Table 2. This is mainly caused by the fact that we do not consider weights for the problem assessed in Table 3.

The average computing time of the algorithms, measured in seconds, confirms that using problem specific heuristics in combination with NSGA-II leads to a smaller computational effort than the application of NSGA-II hybridized with a generic local search approach.

Table 3: Computational Results for $I | inter | ND(TC, L_{max})$

Compare	Algorithm	ONVG	OTNVG	Error	dist	Time
$T=0.2$	NSGA-II-LS	17.500	5.625	0.592	0.001	158
	NSGA-II-H	90.620	67.625	0.265	0.001	12
$T=0.8$	NSGA-II-LS	21.875	17.500	0.213	0.001	162
	NSGA-II-H	30.500	22.875	0.263	0.001	12
$R=0.2$	NSGA-II-LS	18.125	14.000	0.301	0.001	163
	NSGA-II-H	28.125	19.250	0.321	0.002	12
$R=0.8$	NSGA-II-LS	21.250	9.125	0.504	0.001	157
	NSGA-II-H	93.000	71.250	0.206	0.001	13
$n_1=60 / n_2 = 60$	NSGA-II-LS	18.250	8.250	0.496	0.003	90
	NSGA-II-H	47.500	39.250	0.217	0.002	10
$n_1=60 / n_2 = 100$	NSGA-II-LS	9.750	6.250	0.446	0.001	162
	NSGA-II-H	40.250	32.000	0.271	0.002	12
$n_1=100 / n_2 = 60$	NSGA-II-LS	35.750	23.500	0.314	<0.001	151
	NSGA-II-H	80.000	48.750	0.315	0.002	12
$n_1=100 / n_2 = 100$	NSGA-II-LS	15.000	8.250	0.355	0.001	238
	NSGA-II-H	74.500	61.000	0.252	<0.001	15
Overall	NSGA-II-LS	19.688	11.562	0.403	0.001	160
	NSGA-II-H	60.562	45.250	0.264	0.001	12

5 Conclusions and Future Research

In this paper, we considered a NP-hard problem with two interfering job sets where the *TWC* measure is to be minimized for first job set and the L_{max} objective for second one. We were interested in determining an approximation of a true Pareto front using the NSGA-II metaheuristic.

Random key encoding is used to represent the job sequences within NSGA-II. We proved three dominance properties by exploiting the problem structure. NSGA-II is hybridized with a simple generic local search procedure and a local search procedure that is based on these problem specific properties. We conducted experiments based on randomly generated problem instances to compare the performance of both hybridization strategies. It is shown that the hybridization strategy based on problem specific properties outperforms the generic one with respect to solution quality and computational effort.

There are several directions for future research. First of all we are interested in solving Problem (3) using an automated negotiation mechanism. In contrast to the situation in the present paper, each agent has a private objective function when a decentralized problem is considered. The negotiation process is guided by a mediator [9]. We are interested to assessing

the solution quality of the decentralized approach. Therefore, the heuristics presented in this paper assuming complete information can be used.

We believe that it is possible to extend NSGA-II type heuristics to the case of more than two agents. Furthermore, a combination of the two hybridization strategies seems to be fruitful. However, carrying out all the details is part of future research.

References

1. Agnetis, A., Mirchandani, P. B., Pacciarelli, D., Pacifici, A., Nondominated Schedules for a Job-Shop with Two Competing Users. *Computational & Mathematical Organization Theory*, 6(2), 191-217, (2000).
2. Agnetis, A., Pacciarelli, D., Pacifici, A., Multi-agent Single Machine Scheduling. *Annals of Operations Research*, 150(1), 3-15, (2007).
3. Baker, K., Smith, J. C., A Multiple-Criterion Model for Machine Scheduling. *Journal of Scheduling*, 6, 7-16 (2003).
4. Balasubramanian, H., Fowler, J., Keha, A., Pfund, M., Scheduling Interfering Job Sets on Parallel Machines. *European Journal of Operational Research*, 199, 55-67, (2009).
5. Bean, J. C., Genetic Algorithm and Random Keys for Sequencing and Optimization. *ORSA Journal of Computing*, 6(2), 154-160, (1994).
6. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., A Fast and Elitist Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182-197, (2002).
7. Deb, K., Goel, T., A Hybrid Multi-Objective Evolutionary Approach to Engineering Shape Design. *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, 385-399, (2001).
8. Elvikis, D., Hamacher, H. W., T'kindt, V., Scheduling Two Agents on Uniform Parallel Machines with Makespan and Cost Functions. Accepted for Publication in *Journal of Scheduling*, (2011).
9. Fink, A., Decentralized Scheduling Mechanisms: New Concepts. *Proceedings of the VIII Metaheuristics International Conference*, (2009).
10. Jaskiewicz, A., MOMHLIB++: Multiple Objective Metaheuristics Library in C++. <http://home.gna.org/momh/index.html>, (2010).
11. Jaskiewicz, A., Evaluation of Multiple Objective Metaheuristics. *Metaheuristics for Multiobjective Optimization*, Lecture Notes in Economics and Mathematical Systems, 535, 65-89, (2004).
12. Khowala, K., Fowler, J. W., Keha, A., Balasubramanian, H., Single Machine Scheduling with Interfering Job Sets. *Proceedings of the 4th Multidisciplinary International Conference on Scheduling (MISTA) 2009*, 357-365, (2009).
13. Peha, J., Heterogenous-criteria Scheduling: Minimizing Weighted Number of Tardy Jobs and Weighted Completion Time. *Computers & Operations Research*, 22(10), 1089-1100, (1995).
14. Soltani, R., Jolai, F., Zandieh, M., Two Robust Meta-heuristics for Scheduling Multiple Job Classes on a Single Machine with Multiple Criteria. *Expert Systems with Applications*, 37, 5951-5959, (2010).
15. Yuan, J. J., Shang, W. P., Feng, Q., A Note on the Scheduling with Two Families of Jobs. *Journal of Scheduling*, 8, 537-542, (2005).
16. Van Veldhuizen, D. A., Multiobjective Evolutionary Algorithms: Classification, Analyses, and New Innovations. Air Force Institute of Technology Department of Electrical and Computer Engineering, Ohio, (1999).

Scheduling identical jobs on uniform parallel machines to minimize maximum lateness

YK Lin • FY Hsieh • GJ Lin

Abstract

The problem of how to schedule n identical jobs on m uniform parallel machines to minimize some objective function is very common in practice. In many real-world factories, any instance of a product type requires the same processing time on the same machine. Furthermore, factories often buy new equipment but retain their slower, older equipment; this results in machines that have different processing speeds. In this research, we propose a fast linear programming (LP) model to schedule identical jobs on uniform parallel machines to minimize maximum lateness. Moreover, we propose a heuristic to schedule identical jobs with distinct release dates on uniform parallel machines to minimize maximum lateness. Computational results show that the proposed LP model can find optimal solutions in less computational time than other LP models require. The proposed heuristic also outperforms existing heuristics in terms of maximum lateness.

1 Introduction

In this research, we first consider the problem of scheduling n identical jobs on m uniform machines to minimize maximum lateness. Next, we consider an extension of this problem in which jobs are restricted by distinct release dates. Identical jobs all have the same processing time on a given machine because they all have identical processing requirements [1]. In a uniform parallel machine situation, each machine has a different speed (v_i), but each machine works at a consistent rate. Hence, the processing time for j identical jobs numbered 1 to j on machine i in a uniform machine setting can be denoted by $p_{ij} = p_i$. The performance criterion is maximum lateness, which is defined as $L_{\max} = \max_{1 \leq j \leq n} (c_j - d_j)$. It measures the worst violation of the due dates. Following the three-field notation of Graham *et al.* [3], we refer to these two problems as $Q | p_{ij} = p_i | L_{\max}$ and $Q | r_j, p_{ij} = p_i | L_{\max}$ where Q denotes

YK Lin

Department of Industrial Engineering and Systems Management, Feng Chia University
E-mail: yklin@mail.fcu.edu.tw

FY Hsieh

Department of Industrial Engineering and Systems Management, Feng Chia University
E-mail: m9902937@mail.fcu.edu.tw

GJ Lin

Department of Industrial Engineering and Systems Management, Feng Chia University
E-mail: m9745250@fcu.edu.tw

the situation with uniform parallel machines, $p_{ij} = p_i$ indicates that each job j has the same processing time on machine i , r_j is the release date of job j , and the performance measure is maximum lateness.

According to Graham *et al.* [3] problems with uniform machines and unit standard processing times may be formulated as special cases of the transportation problem. $Q | p_j = 1 | \sum f_j$ and $Q | p_j = 1 | f_{\max}$ problems were formulated for the first time as specialized transportation problems in [3]. Dessouky *et al.* [1] developed an LP to find the optimal solution for the $Q | p_j = 1 | C_{\max}$ problem. Blazewicz *et al.* [4] mentioned that $Q | p_j = 1 | C_{\max}$, $Q | p_j = 1 | L_{\max}$, and $Q | p_j = 1 | \sum C_j$ problems may be formulated as special cases of a transportation problem with $O(n^3)$ time complexity. An approach in [2] required $O(m^2)$ time for construction of an optimal schedule for $Q | p_j = 1 | C_{\max}$ problems. Dessouky *et al.* [1] also proposed efficient algorithms for $Q_m | p_j = 1 | C_{\max}$ and $Q | p_j = 1 | \sum C_j$ problems. Both algorithms require $O(n + m \log m)$ time to find the optimal solutions for the $Q_m | p_j = 1 | C_{\max}$ and $Q | p_j = 1 | \sum C_j$ problems. Similarly, Dessouky *et al.* [1] proposed efficient algorithms to solve $Q | p_j = 1 | \sum w_j C_j$, $Q | p_j = 1 | L_{\max}$, $Q | p_j = 1 | \sum w_j U_j$, and $Q | p_j = 1 | \sum T_j$ problems in $O(n \log n)$ time. Moreover, Dessouky *et al.* [1] developed an efficient algorithm that can solve the $Q | p_j = 1 | \max w_j T_j$ problem in $O(n \log n (\log w_{\max} + \log(n/m)))$ time.

For problems in which each job is restricted by an unequal release date r_j , Dessouky *et al.* [1] proposed an algorithm that requires $O(n \log n)$ time to minimize makespan ($Q | r_j, p_j = 1 | C_{\max}$). They also proposed a dynamic programming algorithm that requires $O(mn^{2m+1})$ time to minimize total completion time ($Q | r_j, p_j = 1 | \sum C_j$). Furthermore, Dessouky [5] studied the problem of scheduling n identical jobs with unequal release dates on m parallel uniform machines to minimize maximum lateness ($Q | r_j, p_j = 1 | L_{\max}$). Dessouky first developed an integer programming (IP) model to solve this problem. Then, he proposed six single-pass heuristic procedures to find an approximate solution, and presented a branch-and-bound procedure that optimally solved the problem.

2 LP and heuristic

In this section, we focus on developing a fast LP model and an easily implementable heuristic for $Q | p_{ij} = p_i | L_{\max}$ and $Q | r_j, p_{ij} = p_i | L_{\max}$ problems. The proposed LP model is of theoretical interest to researchers who develop mathematical programming models for identical jobs on uniform parallel machines. The proposed heuristic can find good solutions quickly to meet real world requirements.

Blazewicz *et al.* [4] formulated the $Q | p_j = 1 | L_{\max}$ problem as a transportation problem and solved it to find the optimal solution in $O(n^3)$. In [4] the LP has $2mn^2 + 1$ variables and $mn^2 + mn + n$ constraints. Here we propose a fast LP to solve this problem. First, let the jobs be numbered by the earliest due date (EDD), i.e., $d_1 \leq \dots \leq d_n$. Let x_{ijr} be a 0-1 integer variable that indicates whether job j is assigned to the r th position on machine i . We only consider the cases when $j \geq r$ since all jobs are in EDD order; i.e., we will only assign job 1 to the first position on machine i (corresponding to variable x_{i11}), we will only assign job 2 to

one of the first two positions on machine i (corresponding to variables x_{i21} and x_{i22}), we will only assign job 3 to one of the first three positions on machine i (corresponding to variables x_{i31} , x_{i32} , and x_{i33}), and so on. Sorting the jobs in *EDD* order and matching them with non-decreasing completion times can achieve the optimal solution for the $Q | p_j = 1 | L_{\max}$ problem [1]. The proposed LP model has the following constraints: Constraint (1) states that the objective is to minimize $L_{\max}$. Constraint (2) ensures that each position on each machine can be assigned either one job or no job at all. Constraint (3) ensures that each job can be assigned to one position on one machine. Constraint (4) ensures that $L_{\max}$ is at least as large as each job's lateness, where $p_i r x_{ijr}$ is the completion time of job j if it is processed on the r th position on machine i . Since we only consider the cases when $j \geq r$, the proposed LP has $mn^2 + mn + 1$ variables and $\frac{1}{2}mn^2 + \frac{3}{2}mn + n$ constraints.

$$\text{objective: } \min z = L_{\max} \quad (1)$$

$$\text{subject to: } \sum_{j=1, j \geq r}^n x_{ijr} \leq 1 \quad i = 1, 2, \dots, m \quad r = 1, 2, \dots, n \quad (2)$$

$$\sum_{i=1, r=1, r \leq j}^m \sum_{r=1}^n x_{ijr} = 1 \quad j = 1, 2, \dots, n \quad (3)$$

$$p_i r x_{ijr} - d_j \leq L_{\max} \quad i = 1, 2, \dots, m; j = 1, 2, \dots, n; r = 1, 2, \dots, n; j \geq r \quad (4)$$

$$x_{ijr} \in \{0, 1\} \quad i = 1, 2, \dots, m; j = 1, 2, \dots, n; r = 1, 2, \dots, n; j \geq r \quad (5)$$

Next we propose a heuristic for the $Q | r_j, p_{ij} = p_i | L_{\max}$ problem. We summarize our proposed heuristic as follows.

Step 1. Let U be the set of unscheduled jobs in order of *EDD*, and let t_i be the sum of the processing times of the jobs that have been scheduled on machine i , $i = 1, \dots, m$. Initially, set $U = \{1, \dots, n\}$ and set $t_i = 0$, for $i = 1, \dots, m$.

Step 2. Select the first job j^* from U . For job j^* find machine i^* such that $p_{i^*} = \max \left\{ p_i | \max(r_{j^*}, t_i) + p_i - d_{j^*} \leq 0, 1 \leq i \leq m \right\}$. If i^* cannot be found from the above equation, then conclude that there is no job that can be assigned without being tardy, then select machine i^* such that $i^* = \arg \min_{1 \leq i \leq m} (t_i + p_i - d_{j^*})$

Step 3. Schedule job j^* in the next available position on machine i^* ; update $t_{i^*} = t_{i^*} + p_{i^*}$; set $U = U \setminus \{j^*\}$.

Step 4. Repeat Steps 2-3 until all jobs are scheduled.

3 Computational results

In this section, we present some computational results to demonstrate the performance of the proposed LP model and heuristic. The proposed LP model was coded in AMPL and implemented in CPLEX 11.2; the proposed heuristic was implemented in C. Both LP and heuristic were run on a computer with 3 GHz CPU and 4GB of memory. Processing times $p_{ij} = p_i$ were generated from the uniform distribution [1,100]. We compared our proposed LP for $Q | p_{ij} = p_i | L_{\max}$ with an existing LP proposed in [4]. Due dates were generated in the manner of Potts and Van Wassenhove [6]; i.e., due dates were generated from the uniform

distribution $[P(1-T-R/2), P(1-T+R/2)]$, where $P = \sum_{i=1}^m np_i / m$; T was the average tardiness factor, and R was the relative range of due dates. T and R were set at 0.8 and 0.4. Testing problem sizes were set to $4m20n$, $10m100n$, and $5m200n$. For each problem size, 20 problem instances were randomly generated. Computational results for the proposed LP and the existing LP are given in Table 1. Table 1 shows that both the proposed LP and the existing LP proposed in [4] can find optimal solutions for the $Q | p_{ij} = p_i | L_{\max}$ problem. However, the proposed LP outperformed the existing LP in terms of average computational time for all problem sizes. This result is not surprising since the proposed LP has fewer variables and constraints than the LP proposed in [4].

Table 1 The performance of the proposed LP for the $Q | p_{ij} = p_i | L_{\max}$ problem

Problem size	$4m20n$		$10m100n$		$5m200n$	
	$L_{\max}$	CT	$L_{\max}$	CT	$L_{\max}$	CT
Proposed LP	75.70	0.24	5.35	9.74	106.20	4412.94
Blazewicz <i>et al.</i> [4]	75.70	0.38	5.35	44.74	106.20	8626.96

CT: average computation time in seconds

The result is the average of 20 randomly generated problem instances.

Next, we compared our proposed heuristic for the $Q | r_j, p_{ij} = p_i | L_{\max}$ problem with other existing heuristics proposed by Dessouky [5]. In [5], Dessouky proposed six single-pass heuristics (*FAMR*, *FAMS*, *ECTR*, *ECTS*, *LSTD*, and *LSTF*) to solve the $Q | r_j, p_j = 1 | L_{\max}$ problem. Dessouky [5] indicated that the forward sequencing procedure *ECTS* performs best when the variability in the release times is greater than the variability in the due dates, while the backward sequencing procedure *LSTF* performs best when the variability on the due dates is larger. For the $Q | r_j, p_{ij} = p_i | L_{\max}$ problem, we compared the proposed heuristic with heuristics *ECTS* and *LSTF*. Here, due dates and release dates were generated in a manner similar to that of [7]. We generated ready dates r_j from the uniform distribution $[0, \alpha / m \times (\sum_{i=1}^m np_i / m)]$. In the next step, we generated slack times between due dates and earliest completion time from a uniform distribution $[0, \beta / m \times (\sum_{i=1}^m np_i / m)]$, i.e. $d_j - (r_j + \bar{p})$ where $\bar{p} = \sum_{i=1}^m p_i / m$, α controlled the range of release dates, and β controlled the range of due dates. α and β were set at 0.01, 0.1, 0.25 and 0.5, respectively. For each combination of α , β , and problem size, 20 problem instances were randomly generated. Computational results for *ECTS* and *LSTF*, and for our proposed heuristic, are given in Table 2. Computational results show that our proposed heuristic outperformed *ECTS* and *LSTF* in terms of maximum lateness. For example, for the $10m500n$ problem instances, *ECTS* deviated 73.70% from the proposed heuristic and *LSTF* deviated 139.90% from the proposed heuristic. The average computational times for the $10m500n$ problem instances were 0.001125(s), 0.000144(s), and 0.002544(s) for the proposed heuristic, *ECTS*, and *LSTF*, respectively.

Table 2 The performance of the heuristics for the $Q | r_j, p_{ij} = p_i | L_{\max}$ problem

(Existing heuristic- proposed heuristic)/ proposed heuristic							
Problem size		$4m20n$		$10m100n$		$10m500n$	
α	β	<i>ECTS</i>	<i>LSTF</i>	<i>ECTS</i>	<i>LSTF</i>	<i>ECTS</i>	<i>LSTF</i>
0.01	0.01	0.0000	0.0018	0.0000	0.0062	0.0004	0.2786
0.01	0.10	0.0046	0.0598	0.0265	0.1415	0.1417	0.3571
0.01	0.25	0.0643	0.2888	0.3612	0.4803	0.5765	0.4892
0.01	0.50	0.4961	1.0078	1.8321	3.5016	2.5024	1.6538

0.10	0.01	0.0004	0.0028	0.0028	0.0098	0.0060	0.3621
0.10	0.10	0.0477	0.0550	0.1465	0.1378	0.1540	0.4995
0.10	0.25	0.2288	0.1983	0.5400	0.3986	0.6030	0.6453
0.10	0.50	0.9851	0.8425	2.4965	1.8724	2.4241	1.9321
0.25	0.01	0.0000	0.0382	0.0026	0.0998	0.0093	0.7758
0.25	0.10	0.0390	0.0735	0.1602	0.1341	0.1649	0.7531
0.25	0.25	0.1956	0.2630	0.5413	0.3967	0.5625	1.0832
0.25	0.50	1.1685	0.9433	2.4055	1.5213	2.0792	2.4978
0.50	0.01	0.0000	0.1246	0.0008	0.6923	0.0160	2.3292
0.50	0.10	0.0696	0.1710	0.1112	0.5097	0.2045	2.5303
0.50	0.25	0.1530	0.1855	0.6140	0.7049	0.6013	3.0122
0.50	0.50	0.9831	0.8208	1.4267	0.7147	1.7469	3.1840
Average		0.2772	0.3173	0.6667	0.7076	0.7370	1.3990
The result is the average of 20 randomly generated problem instances.							

4 Conclusions

In this research, we proposed an LP model for scheduling identical jobs on uniform parallel machines to minimize maximum lateness. Also, we proposed a heuristic for scheduling identical jobs with distinct release dates on uniform parallel machines to minimize maximum lateness. Computational results have shown that the proposed LP model can find optimal solutions with less computational time than an earlier LP model. Also, the proposed heuristic outperforms other existing heuristics in terms of maximum lateness.

Acknowledgement: This work has been supported by the National Science Council of Taiwan, Republic of China, under grant number NSC 99-2221-E-035-038.

References

- [1] Dessouky MI, Lageweg BJ, Lenstra JK and van de Velde SL. Scheduling identical jobs on uniform parallel machines. *Statistica Neerlandica*, 44:115-123(1990).
- [2] Blazewicz J, Ecker K, Pesch E, Schmidt G, and Weglarz J. *Scheduling Computer and Manufacturing Processes*, Springer Verlag, Berlin, New York, 1996.
- [3] Graham RL, Lawler EL, Lenstra JK and Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 1979;(5):287-326.
- [4] Blazewicz J, Dror M, and Weglarz J. Mathematical programming formulations for machine scheduling: a survey. *European Journal of Operational Research* 1991; (51): 283-300.
- [5] Dessouky MM. Scheduling identical jobs with unequal ready times on uniform parallel machines to minimize maximum lateness. *Computers and Industrial Engineering* 1998;(34): 793-806.
- [6] Potts CN and Van Wassenhove LN. A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters* 1982;(1):177-81.
- [7] Mönch L, Balasubramanian H, Fowler JW, and Pfund ME. Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers and Operations Research* 2005; (32): 2731-2750.

Generating Better Cyclic Fair Sequences Faster with Aggregation

Jeffrey W. Herrmann

Abstract Fair sequences allocate capacity to competing demands in a variety of manufacturing and computer systems. This paper considers three problems that motivate generating cyclic fair sequences: (1) the response time variability problem, (2) routing jobs to parallel servers (the waiting time problem), and (3) finding balanced words. This paper discusses the similarities among these three problems and presents a general aggregation approach. Computational results show that using aggregation with stride scheduling both generates solutions that are more fair and reduces computational effort. The paper concludes with some ideas for applying the approach to related problems.

1 Introduction

In many applications, it is important to schedule the resource's activities in some fair manner so that each demand receives a share of the resource that is proportional to its demand relative to the competing demands. This problem arises in a variety of domains. A mixed-model, just-in-time assembly line produces different products at different rates. Maintenance and housekeeping staff need to perform preventive maintenance and housekeeping activities regularly, but some machines and areas need more attention than others. Vendors who manage their customers' inventory must deliver material to their customers at rates that match the different demands. Asynchronous transfer mode (ATM) networks must transmit audio and video files at rates that avoid poor performance.

The idea of fair sequences occurs in many different areas, including those mentioned above. Kubiak [1] reviewed the need for fair sequences in different domains and concluded that there are multiple definitions of a "fair sequence." This review discussed results for multiple related problems, including the product rate variation problem, generalized pinwheel scheduling, the hard real-time periodic scheduling problem, the periodic maintenance scheduling problem, stride scheduling, minimizing response time variability (RTV), and peer-to-peer fair scheduling. Bar-Noy et al. [2] discussed the generalized maintenance scheduling problem, in which the objective is to minimize the long-run average service and operating cost per time slot. Bar-Noy et al. [3] considered the problem of creating a schedule in which a single broadcast channel transmits each page perfectly periodically (that is, the interval between consecutive transmissions of a page is always the same). The goal is to find a feasible schedule so that the lengths of the scheduled periods are close to the ideal periods that will minimize average waiting time.

Jeffrey W. Herrmann
University of Maryland, College Park
E-mail: jwh2@umd.edu

In the periodic maintenance scheduling problem, the intervals between consecutive completions of the same task must equal a predetermined quantity. Wei and Liu [4] presented sufficient conditions for the existence of a feasible solution for a given number of servers. They also suggested that machines with the same maintenance interval could be replaced by a substitute machine with a smaller maintenance interval and that this replacement would facilitate finding a feasible solution. This concept, which was not developed into a solution algorithm, is similar to the aggregation proposed here. Park and Yun [5] and Campbell and Hardin [6] studied other problems that require the activities to be done strictly periodically and developed approaches to minimize the number of required resources. Park and Yun considered activities such as preventive maintenance tasks that take one time unit but can have different periods and different resource requirements. Campbell and Hardin studied the delivery of products to customers, where each delivery takes one day but the customers have different replenishment periods that must be satisfied exactly.

Waldspurger and Weihl [7] considered the problem of scheduling multithreaded computer systems. In such a system, there are multiple clients, and each client has a number of tickets. A client with twice as many tickets as another client should be allocated twice as many quanta (time slices) in any given time interval. Waldspurger and Weihl introduced the stride scheduling algorithm to solve this problem. They also presented a hierarchical stride scheduling approach that uses a balanced binary tree to group clients, uses stride scheduling to allocate quanta to the groups, and then, within each group, uses stride scheduling to allocate quanta to the clients. Although they note that grouping clients with the same number of tickets would be desirable, their approach does not exploit this. Indeed, the approach does not specify how to create the binary tree.

Kubiak [1] showed that the stride scheduling algorithm is the same as Jefferson's method of apportionment and is an instance of the more general parametric method of apportionment [8]. Thus, the stride scheduling algorithm can be parameterized.

The problem of mixed-model assembly lines has the same setup as the RTV problem but a different objective function. Miltenburg [9] considered objective functions that measure (in different ways) the total deviation of the actual cumulative production from the desired cumulative production and presented an exact algorithm and two heuristics for finding feasible solutions. Inman and Bulfin [10] considered the same setting but assigned due dates to each and every unit of the products. They then presented the results of a small computational study that compared earliest due date (EDD) schedules to those generated using Miltenburg's algorithms on a set of 100 instances. They show that the EDD schedules have nearly the same total deviation but can be generated much more quickly.

A more flexible approach in a cyclic situation is to minimize the variability in the time between consecutive allocations of the resource to the same demand (the same product or client, for instance). Corominas et al. [11] presented and analyzed the RTV objective function (which will be defined precisely below), showed that the RTV problem is NP-hard, and presented a dynamic program and a mathematical program for finding optimal solutions. Because those approaches required excessive computational effort, they conducted experiments to evaluate the performance of various heuristics. However, the Webster and Jefferson heuristics (described below) reportedly performed poorly across a range of problem instances. Garcia et al. [12] presented metaheuristic procedures for the problem. Kubiak [13] reviews these problems and others related to apportionment theory.

Unlike previous work on aggregation and cyclic fair sequences (reviewed in Section 2), this paper considers multiple measures of fairness and shows how using aggregation improves all of them.

2 Previous Work on Aggregation

Aggregation is a well-known and valuable technique for solving optimization problems, especially large-scale mathematical programming problems. Model aggregation replaces a large optimization problem with a smaller, auxiliary problem that is easier to solve [14]. The

solution to the auxiliary model is then disaggregated to form a solution to the original problem. Model aggregation has been applied to a variety of production and distribution problems, including machine scheduling problems. In the context of cyclic fair sequences, aggregation combines products with equal demand. Although concepts similar to those of aggregation have been mentioned previously [4], the following papers are the first, in the context of cyclic fair sequences, to study the benefits of aggregation in a systematic way.

Herrmann [15] described the RTV problem and presented a heuristic that combined aggregation and parameterized stride scheduling. The aggregation approach combines products with the same demand into groups, creates a sequence for those groups, and then disaggregates the sequence into a sequence for each product.

For the single-server RTV problem, Herrmann [16] precisely defined the aggregation approach and described the results of extensive computational experiments using the aggregation approach in combination with the heuristics presented by [11]. The results showed that solutions generated using the aggregation approach can have dramatically lower RTV than solutions generated without using the aggregation approach. Herrmann [17] considered a more sophisticated “perfect aggregation” approach that can generate zero RTV sequences for some instances.

Herrmann [18] considered the RTV problem when multiple servers, working in parallel, are available, and presented a specific aggregation approach. The results showed that, in most cases, combining aggregation with other heuristics does dramatically reduce both RTV and computation time compared to using the heuristics without aggregation.

Herrmann [19] studied the problem of routing deterministic arriving jobs to parallel servers with deterministic service times, when the job arrival rate equals the total service capacity. This is known as the waiting time problem (WTP). This requires finding a periodic routing policy. The paper proposed an aggregation approach, and computational experiments showed that using aggregation not only reduces average waiting time but also reduces computational effort.

Herrmann [20] considered the balanced word problem (BWP), in which a set of letters must be assigned to positions in a sequence so that the letters occur at specified densities. Two different balance measures were used. The work used an aggregation approach in combination with different heuristics. The results of computational experiments showed that using the stride scheduling heuristic with aggregation generates the best solutions with the least computational effort (compared to the other heuristics).

3 Problem comparison

The RTV problem, WTP, and BWP have different objective functions but many similarities. All are concerned with finding a fair sequence. In a perfectly fair sequence, the positions assigned to one product (in the RTV problem), one server (in the WTP), or one letter (in the BWP) would be spaced perfectly evenly. The RTV would equal zero, jobs would never wait, and all of the letters would be balanced. But the discrete nature of the positions and the need to allocate positions to multiple products (servers, letters) may make this impossible.

The aggregation schemes presented in [16, 19, 20] are essentially the same. (The aggregation scheme in [18] adds an additional constraint to accommodate the multiple servers.) The similarity depends upon the similarity of the underlying problems, which becomes clear once we establish a “translation” between the terms used to describe the components of the problems, as shown in Table 1.

Table 1. Comparison of terminology for balanced word, RTV, and job routing problems.

Problem component	Balanced word problem	RTV problem	Waiting time problem
Object being assigned to positions in the sequence	Letter	Product	Server
Frequency that must be satisfied	Density	Demand	Service rate
Position in the sequence	A position in a word	A time slot for processing by the server	An arriving job

4 Problem Formulation

Given the translation above, we can formulate a general cyclic fair sequencing problem. We are given a set of n objects, each with a rational frequency p_i . The sum of the frequencies equals 1. Then, there exists a positive integer T and positive integers $x_1, \dots, x_n$ such that $p_i = x_i / T$ for $i = 1, \dots, n$ and $\gcd(x_1, \dots, x_n) = 1$. Thus, $x_1 + \dots + x_n = T$. Hereafter, we will describe an instance by the values of $(x_1, \dots, x_n)$, with $x_1 \geq x_2 \geq \dots \geq x_n$. We call each value the “count” of the corresponding object.

Given an instance $(x_1, \dots, x_n)$, a sequence s of length T is a feasible sequence if exactly one object is assigned to each and every one of the T positions and each and every object i occurs exactly x_i times in s .

The sequence will be repeated to make an infinite sequence. We seek a sequence that is as “fair” as possible. Different measures of fairness have been proposed, motivated by different applications.

The first two measures are related to the balanced word problem (BWP). The count balance measure considers how often a particular object appears in a subsequence. A sequence is c -balanced if, for any two subsequences x and y of the same size (number of positions), then $||x|_i - |y|_i| \leq c$, where $|x|_i$ is the number of times that i occurs in the factor x [13]. We will define the *count balance* of a sequence as the minimal such value of c . For example, the count balance of $(1231211321)^\infty$ equals 2 because $|11|_1 - |23|_1 = 2$ and $||x|_i - |y|_i| \leq 2$ for all subsequences x and y and all objects i .

The gap balance considers the gaps between consecutive occurrences of an object. Given a sequence, an object a is m -balanced if, whenever there exists an a -chain aWa in the sequence, any subsequence W' such that $|W'| = |W| + m + 1$ satisfies $|W'|_a \geq |W|_a + 1$. The sequence is m -balanced if each object is m -balanced [21]. We will define the *gap balance* of a sequence as the minimal such value of m . For example, in $(313132)^\infty$, the gap balance of the objects 2 and 3 equals 0, and the gap balance of the object 1 equals 2. Note that the subsequence 3 in the 1-chain 131 is 2 positions shorter than 323, the longest subsequence with no instance of the object 1. Therefore, the gap balance of this word equals 2.

Thus, we can describe BWP-count (and BWP-gap) as follows: Given an instance $(x_1, \dots, x_n)$, find a finite word S of length T that minimizes the count balance (gap balance) of the infinite word U that is the infinite repetition of S subject to the constraints that exactly one object is assigned to each position of S and object i occurs exactly x_i times in S for $i = 1, \dots, n$.

The complexity of BWP-count appears to be open. Given an instance, finding a word with a count balance that equals 1 requires finding a regular word. The complexity of this problem is open [13]. Likewise, the complexity of BWP-gap appears to be open. Given an instance, finding a word with an gap balance that equals 0 requires finding a constant gap word for $(x_1, \dots, x_n)$. The complexity of the constant gap problem is open [1]. Nevertheless, these problems are related to the Periodic Maintenance Scheduling Problem, which is NP-complete in the strong sense [13], and the RTV problem, which is NP-hard [11].

The RTV of a feasible sequence equals the sum of the RTV for each object. If object i occurs at positions $\{p_{i1}, \dots, p_{id_i}\}$, its RTV is a function of the intervals between each position, which are $\{\Delta_{i1}, \dots, \Delta_{id_i}\}$, where the intervals are measured as $\Delta_{ik} = p_{ik} - p_{i,k-1}$ (with $p_{i0} = p_{id_i} - T$). The average interval for object i is T/x_i , so we calculate RTV as follows:

$$RTV = \sum_{i=1}^n \sum_{k=1}^{x_i} \left(\Delta_{ik} - \frac{T}{x_i} \right)^2$$

In the waiting time problem (WTP), there is a queueing system that consists of arriving jobs and a set of n parallel servers, each with its own queue. Starting at $t = 0$, one job arrives every time unit. Every arriving job must be routed to one of the servers at the moment that it arrives. If the server is idle, then the job immediately begins processing. If the server is busy, then the job waits in that server's queue. Server i has a service rate of p_i jobs per time unit, so the job processing time is a fixed $1/p_i$ time units. The sequence specifies the servers to which the jobs should be routed.

The objective is to minimize the long-run average waiting time W . There exist periodic policies that minimize W , and T is the smallest period of such optimal periodic policies [22].

To simplify the evaluation of a policy, define u_i^s as the total amount of time units that server i has been idle between time $t = 0$ and $t = s$. This depends upon how many jobs were routed to server i , when they arrived, and the server's processing time. Then, S^t , the total unused work capacity at time t , and S , the total unused work capacity, can be determined as follows:

$$\begin{aligned} S^t &= \sum_{i=1}^n a_i u_i^t \\ d_i &= \lim_{t \rightarrow \infty} a_i u_i^t \\ S &= \lim_{t \rightarrow \infty} S^t = \sum_{i=1}^n d_i \end{aligned}$$

When a periodic policy is applied, $W = S^t - (n-1)/2$ for $t \geq T-1$ (van der Laan, 2005).

The complexity of WTP also appears to be open. Given an instance of WTP, the question of finding a periodic policy with $W = 0$ requires finding a constant gap word for $(x_1, \dots, x_n)$.

Consider, as an example, the following three-object instance: $(x_1, x_2, x_3) = (4, 3, 2)$ and $T = 9$. Consider the sequence $(112231123)^\infty$. The count balance of this sequence equals 2, and the gap balance equals 3 (because the gap balance of the object 1 equals 3). The RTV of this sequence equals 13.25. The average waiting time $W = 7/9$.

Now, consider the sequence $(121312123)^\infty$. The count balance of this sequence also equals 2 (because $|212|_2 - |131|_2 = 2$), but the gap balance equals 2 (because the gap balance of the object 2 equals 2). The RTV of this sequence equals 3.25. The average waiting time $W = 4/9$. Clearly, this sequence is more fair than the first one.

5 Aggregation

The aggregation approach used here repeatedly aggregates a set of objects until it cannot be aggregated any more. Each aggregation combines objects that have the same count into a group. These objects are removed, and the group becomes a new object in the new aggregated alphabet.

The key distinctions between the hierarchical stride scheduling algorithm of [7] and the aggregation approach presented here are (1) the hierarchical stride scheduling algorithm requires using the stride scheduling algorithm to disaggregate each group, since the objects in a group may have unequal counts and (2) the placement of objects in the tree is not specified. Because our aggregation scheme groups objects with equal counts, the disaggregation is much simpler. The limitation, however, is that the instance must have some equal count objects.

Also, unlike [3], our aggregation approach does not adjust the demands so that they fit into a single weighted tree (which would lead to a perfectly periodic schedule). Instead, the demands are fixed, and the aggregation creates a forest of weighted trees, which are used to disaggregate a sequence.

The aggregation procedure generates a sequence of instances $I_0, \dots, I_H$. (H is the index of the last aggregation created.) The aggregation can be done at most $n-1$ times because the number of objects decreases by at least one each time an aggregation occurs. Thus $H \leq n-1$. Aggregation runs in $O(n^2)$ time because each aggregation requires $O(n)$ time and there are at most $n-1$ aggregations.

The notation used in the algorithm that follows enables us to keep track of the aggregations in order to describe the disaggregation of a sequence precisely. Let I_0 be the original instance and I_k be the k -th instance generated from I_0 . Let n_k be the number of objects in instance I_k . Let B_j be the set of objects that form the new object j , and let $B_j(i)$ be the i -th object in that set. As the aggregation algorithm is presented, we describe its operation on the following five-object example: $I_0 = (3, 2, 2, 1, 1)$, $n = 5$, and $T = 9$.

Aggregation. Given: an instance I_0 with counts $(x_1, x_2, \dots, x_n)$.

1. Initialization. Let $k = 0$ and $n_0 = n$.
2. Stopping rule. If all of the objects in I_k have different counts, return I_k and $H = k$ because no further aggregation is possible. Otherwise, let G be the set of objects with the same count such that any smaller count is unique.

Example. With $k = 0$, $G = \{4, 5\}$ because $x_4 = x_5$.

3. Aggregation. Let $m = |G|$ and let i be one of the objects in G . Create a new object $n + k + 1$ with count $x_{n+k+1} = mx_i$. Create the new instance I_{k+1} by removing from I_k all m objects in G and adding object $n + k + 1$.
1. Set $B_{n+k+1} = G$. $n_k = n_{k-1} - m + 1$. Increase k by 1 and go to Step 2.

Example. With $k = 0$ and $G = \{4, 5\}$, the new object 6 has count $x_6 = 2 \times 1 = 2$. $B_6 = \{4, 5\}$. The objects in I_1 are $\{1, 2, 3, 6\}$. When $k = 1$, $G = \{2, 3, 6\}$. The new object 7 has count $x_7 = 3 \times 2 = 6$, and $B_7 = \{2, 3, 6\}$. The objects in I_2 are $\{1, 7\}$, which have different counts. Table 2 describes the instances created for this example.

Table 2. The counts for the five original objects in the example instance I_0 and the two new objects in the aggregate instances I_1 and I_2 .

	x_1	x_2	x_3	x_4	x_5	x_6	x_7
I_0	3	2	2	1	1		
I_1	3	2	2			2	
I_2	3						6

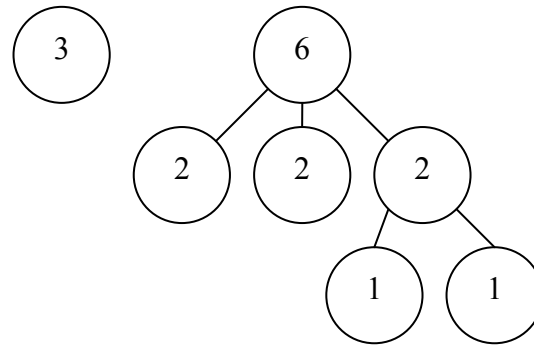


Figure 1. The forest corresponding to the aggregation of the example. The five leaf nodes correspond to the original objects in the example. The two parent nodes correspond to the new objects created during the aggregation. The two root nodes correspond to the objects remaining in the most aggregated instance.

At any point during the aggregation, the sum of the counts in a new instance will equal T because the count of the new object equals the sum of the counts of the objects that were combined to form it. We can represent the aggregation as a forest of weighted trees. There is one tree for each object in the aggregated instance I_H . The weight of the root of each tree is the sum of the counts of the objects in I_0 that were aggregated to form the corresponding object in I_H . The weight of any node besides the root node is the weight of its parent divided by the number of children of the parent. The leaves of a tree correspond to the objects in I_0 that were aggregated to form the corresponding object in I_H , and each one's weight equals the count of that object. The forest has one parent node for each new object formed during the aggregation, and the total number of nodes in the forest equals $n + H < 2n$. Figure 1 shows the forest corresponding to the aggregation of the (3, 2, 2, 1, 1) instance.

6 Disaggregation

When aggregation is complete, we must find a feasible periodic sequence for the aggregated instance I_H and then disaggregate that sequence. This section presents the disaggregation procedure.

Let S_H be a feasible sequence for the instance I_H . In particular, S_H is a sequence of length T . Each position in S_H is a object in the instance I_H . Disaggregating S_H requires H steps that correspond to the aggregations that generated the instances I_1 to I_H , but they will, naturally, be considered in reverse order. We disaggregate S_H to generate S_{H-1} and then

continue to disaggregate each sequence in turn to generate $S_{H-2}, \dots, S_0$. S_0 is a feasible sequence for I_0 , the original instance.

The basic idea of disaggregating a sequence S_k is to replace each new object with the objects used to form it. Object $n+k$ was formed to create instance I_k from the objects in B_{n+k} , which were in I_{k-1} . It has x_{n+k} positions in S_k . According to the aggregation scheme, $x_{n+k} = mx_i$, where $m = |B_{n+k}|$ and i is one of the objects in B_{n+k} . The first position in S_k assigned to object $n+k$ will, in the new sequence S_{k-1} , go to the first object in B_{n+k} , the second position assigned to object $n+k$ will go to the second object in B_{n+k} , and so forth. This will continue until all x_{n+k} positions have been assigned. Each object in B_{n+k} will get x_{n+k}/m positions in S_{k-1} .

In the following algorithm, $j = S_k(a)$ means that object j is in position a in sequence S_k , and $B_{n+k}(i)$ is the i -th object in B_{n+k} .

Disaggregation. Given: The instances $I_0, \dots, I_H$ and the sequence S_H , a feasible sequence for the instance I_H .

1. Initialization. Let $k = H$.
2. Set $m = |B_{n+k}|$ and $i = 1$.
3. For $a = 0, \dots, T-1$, perform the following step:
 - a. If $S_k(a) < n+k$, assign $S_{k-1}(a) = S_k(a)$. Otherwise, assign $S_{k-1}(a) = B_{n+k}(i)$, increase i by 1, and, if $i > m$, set $i = 1$.
4. Decrease k by 1. If $k > 0$, go to Step 2. Otherwise, stop and return S_0 .

Example. Consider the aggregation of the instance $(3, 2, 2, 1, 1)$ presented earlier and the sequence $S_2 = (771771771)^\infty$, which is a feasible sequence for the aggregated instance I_2 . When $k = 2$, $n+k = 7$, and $B_7 = \{2, 3, 6\}$. The positions in S_2 that are assigned to server 7 will be reassigned to servers 2, 3, and 6. The resulting sequence $S_1 = (231621361)^\infty$.

When $k = 1$, $n+k = 6$, and $B_6 = \{4, 5\}$. The positions in S_1 that are assigned to object 6 will be reassigned to objects 4 and 5. The resulting sequence $S_0 = (231421351)^\infty$. Table 3 lists these three sequences.

As noted earlier, there are at most $n-1$ aggregations. Because each sequence disaggregation requires $O(T)$ effort, disaggregation runs in $O(nT)$ time in total.

Table 3. The disaggregation of sequence S_2 for instance I_2 in the example. The first row is S_2 , a feasible sequence for instance I_2 . The second row is S_1 , a feasible sequence for instance I_1 . The third row is S_0 , a feasible sequence for instance I_0 .

a	0	1	2	3	4	5	6	7	8
$S_2(a)$	7	7	1	7	7	1	7	7	1
$S_1(a)$	2	3	1	6	2	1	3	6	1
$S_0(a)$	2	3	1	4	2	1	3	5	1

7 Stride Scheduling

Studies of the RTV problem, WTP, and BWP have proposed various heuristics because finding optimal solutions is computationally expensive. The stride scheduling algorithm [7] has been used repeatedly and will be discussed here.

The parameterized stride scheduling algorithm builds a fair sequence and performs well at minimizing the maximum absolute deviation [1]. The algorithm has a single parameter δ that can range from 0 to 1. This parameter affects the relative priority of low-demand products and their absolute position within the sequence. When δ is near 0, low-demand products will be positioned earlier in the sequence. When δ is near 1, low-demand products will be positioned later in the sequence.

Parameterized stride scheduling. Given: an instance $(x_1, \dots, x_n)$ and the parameter δ .

1. Initialization. $m_{i0} = 0$ for $i = 1, \dots, n$. (m_{ik} is the number of times that product i occurs in the first k positions of the sequence.)
2. For $k = 0, \dots, T-1$, perform the following steps:
 - a. Assign position $k+1$ to the lowest-numbered object i^* such that

$$i^* = \arg \max_i \left\{ \frac{x_i}{m_{ik} + \delta} \right\}$$

- b. Set $m_{i^*,k+1} = m_{i^*,k} + 1$. For all j not equal to i^* , set $m_{j,k+1} = m_{jk}$.

The computational effort of the algorithm is $O(nT)$. For the five-object instance introduced earlier, the parameterized stride scheduling algorithm (with $\delta = 0.5$) generates the sequence $(123145231)^\infty$.

Two versions of the parameterized stride scheduling algorithm are the well-known “Webster” and “Jefferson” apportionment methods [11]. These names refer to the traditional parametric methods of apportionment to which they are equivalent [1, 8]. The Webster heuristic uses $\delta = 0.5$. The Jefferson heuristic uses $\delta = 1$.

8 The Benefits of Aggregation

Experiments with thousands of instances of the BWP, RTV problem, and WTP have shown that using aggregation in combination with good heuristics both generates better sequences and reduces computational effort. In particular, using aggregation with parameterized stride scheduling algorithm generates good results.

The comparison of stride scheduling with other problem-specific heuristics was reported in [16, 18, 19, 20]. This paper discusses how aggregation with stride scheduling performs across a range of metrics of fairness. In addition, we will not consider the multiple-server RTV problem because the WTP and BWP have only a single sequence.

To generate the instances used in the computational study, we used the following scheme. First, we set the value of T and the number of objects n . Then, for this combination, we generated $T - n$ random numbers from a discrete uniform distribution over $\{1, \dots, n\}$. We then let x_i equal one plus the number of copies of i in the set of $T - n$ random numbers (this avoided the possibility that any $x_i = 0$). We generated 100 instances for each of 18 combinations of T and n shown in Table 4. All of the generated instances can be aggregated. (Note, in [16, 18, and 19], more instances were generated and used for testing aggregation.)

Table 4. Combinations of T and n used to generate instances.

T	n								
100	10	20	30	40	50	60	70	80	90
500	50	100	150	200	250	300	350	400	450

Table 5. Average number of aggregations for the instances in each problem set.

T	n	Average number of aggregations
100	10	2.66
	20	6.00
	30	5.71
	40	5.11
	50	4.03
	60	3.68
	70	3.23
	80	2.64
	90	2.07
500	50	10.77
	100	9.20
	150	7.39
	200	6.09
	250	5.10
	300	4.34
	350	3.84
	400	3.20
	450	2.69

For each instance, we constructed solutions as follows. First, we applied the stride scheduling algorithm (with $\delta = 0.5$ and with $\delta = 1$) to the instance (we call this the H solution). Next, we aggregated the instance. For the aggregate instance, we applied the heuristic to construct an aggregated solution. We disaggregated this solution to construct the AHD solution.

When considering the RTV measure, we also applied the exchange heuristic to the heuristic solution to construct the HE sequence. Finally, we applied the exchange heuristic to the AHD sequence to construct the AHDE sequence. The exchange heuristic significantly reduces the RTV of the solutions generated by the heuristics (without aggregation) [11]. Herrmann [16] describes the exchange heuristic in detail.

When considering the waiting time measure, the special greedy (SG) algorithm was used to improve an existing policy. Given a feasible solution, we apply the SG algorithm repeatedly until it converges. Applying the SG algorithm to the heuristic solution generated the HG sequence. Applying the SG algorithm to the AHD sequence generated the AHDG sequence. Under some conditions, the SG algorithm will generate an optimal solution [22]. Herrmann [19] describes the SG algorithm in detail.

Here we report on stride scheduling with $\delta = 0.5$ (this is also known as the Webster heuristic). The results with $\delta = 1$ were very similar. (For details, see [16, 19, 20].)

Before discussing the results of the heuristics, we consider first how many times that an instance could be aggregated. Table 5 shows that the average number of aggregations almost always decreases as n increases. When $T = 100$, the average number of aggregations increases as n increases from 10 to 20. The average number of aggregations per instance is near six for $T = 100$ and $n = 20$, but, as n increases, this decreases to just over two.

As n approaches T , the average number of objects in the aggregated instances also decreases because the aggregation depends upon the number of distinct values of x_i . Each distinct value leads to an aggregation of multiple objects and generates a new object in the aggregated instance. Thus, the number of objects in the aggregated instance generally equals the number of aggregations needed to create it. Of course, there are some cases in which two new objects can be combined, which increases the number of aggregations and reduces the

number of objects, and some objects may have unique values, but this occurred less often as n increased. As n approaches T , the number of distinct values decreases, so there are fewer aggregations and fewer objects in the aggregated instances.

Tables 6, 7, 8, and 9 list the results of the experiments. The measure (count balance, gap balance, RTV, or waiting time) was averaged over the 100 instances in each problem set.

We will first consider the results for minimizing the count balance. As shown in Table 6, using aggregation with stride scheduling reduced the average count balance. As n increased, there is a trend towards smaller count balances (because there are more objects that have counts near 1).

As shown in Table 7, the gap balance can be quite large for solutions generated using stride scheduling as n increased. In these cases, the large gap balances occur because the small-count objects are all placed together, and multiple copies of the larger-count objects are bunched together, which increases the gap balance of the larger-count objects. Using aggregation with stride scheduling significantly reduced the average gap balance, and the solutions were more balanced as n increased because aggregation combined the objects into a small number of high-count objects that were distributed more evenly.

On the RTV performance measure, the results in Table 8 show that the H sequences may perform extremely poorly for large values of n . The AHD sequences are much better, especially as n increased. As expected, using the exchange heuristic generated better sequences, but aggregation is still useful. Compared with the HE sequences, the AHDE sequences reduce RTV dramatically when n is moderate (but not at the extremes). When n is near T , the AHDE sequences are about the same as the HE sequences because the exchange heuristic is very good at constructing low-RTV sequences in those cases.

As shown in Table 9, the average waiting time in the H sequences generated by stride scheduling are quite large for all values of n . The SG algorithm, which constructs the HG policies, reduces average waiting time dramatically when improving these sequences, especially as n increased. The quality of the AHD and AHDG sequences is very good compared to the H and HG sequences. The stride scheduling algorithm, which generates poor-quality sequences by itself, performs well when used with aggregation. When n is near T , all of the AHD sequences have practically no waiting. Compared to the AHD sequences, the AHDG sequences are slightly better. In general, there is little room for improvement because the AHD sequences are quite good.

Table 6. Average values of the count balance for the H and AHD sequences generated using stride scheduling (with $\delta = 0.5$) and aggregation.

T	n	H	AHD
100	10	2.70	2.01
	20	2.85	2
	30	3	2
	40	3.04	1.99
	50	2.80	1.95
	60	3.24	1.82
	70	3.37	1.63
	80	3.08	1.58
	90	2.43	1.36
500	50	3	2
	100	3	2
	150	3.02	2
	200	3.54	2
	250	3.71	2
	300	3.68	1.95
	350	3.79	1.92
	400	3.65	1.71
	450	2.96	1.42

Table 7. Average values of the gap balance for the H and AHD sequences generated using stride scheduling (with $\delta = 0.5$) and aggregation.

T	n	H	AHD
100	10	8.55	4.57
	20	16.04	3.97
	30	23.70	3.32
	40	29.29	2.89
	50	34.20	2.65
	60	49.46	2.16
	70	64.16	1.80
	80	75.74	1.58
	90	83.80	0.79
500	50	45.31	7.28
	100	84.33	5.89
	150	117.60	5.09
	200	150.47	4.16
	250	179.02	3.49
	300	249.33	2.91
	350	322.94	2.52
	400	388.71	1.92
	450	443.66	1.34

Table 8. Average values of RTV for the H, HE, AHD, and AHDE sequences generated using stride scheduling (with $\delta = 0.5$), aggregation, and the exchange heuristic.

T	n	H	HE	AHD	AHDE
100	10	262.5	98.3	95.9	73.0
	20	1,016.6	165.7	82.4	59.1
	30	2,548.8	180.2	60.7	39.0
	40	4,662.1	217.6	47.8	26.1
	50	8,540.9	85.6	39.5	18.3
	60	16,072.1	36.3	25.9	9.0
	70	26,778.2	5.3	14.0	3.7
	80	35,352.1	1.4	10.1	1.3
	90	31,222.4	0.3	1.8	0.3
500	50	22,159.6	2,006.0	862.8	513.6
	100	111,484.2	2,099.7	590.2	306.3
	150	285,794.8	6,778.6	434.8	211.6
	200	561,918.5	1,850.0	315.3	153.0
	250	1,056,653.4	650.9	212.8	83.0
	300	2,012,186.5	415.9	152.9	42.1
	350	3,358,079.2	25.1	102.3	17.7
	400	4,421,555.9	7.7	51.0	6.5
	450	3,908,763.9	1.5	20.6	1.5

Table 9. Average values of waiting time for the H, HG, AHD, and AHDG sequences generated using stride scheduling (with $\delta = 0.5$), aggregation, and the SG algorithm.

T	n	H	HG	AHD	AHDG
100	10	1.70	1.25	1.00	0.94
	20	2.65	1.59	0.70	0.65
	30	3.56	1.88	0.48	0.44
	40	4.58	1.91	0.36	0.32
	50	5.65	1.99	0.27	0.25
	60	6.72	1.28	0.18	0.16
	70	7.10	0.60	0.10	0.09
	80	6.34	0.22	0.06	0.06
	90	4.04	0.10	0.01	0.01
500	50	7.49	4.91	1.35	1.20
	100	12.88	6.23	0.85	0.71
	150	17.16	8.60	0.59	0.50
	200	22.64	8.90	0.42	0.37
	250	28.64	7.59	0.29	0.25
	300	33.85	4.95	0.20	0.18
	350	35.57	1.96	0.13	0.12
	400	31.73	0.34	0.06	0.06
	450	20.21	0.10	0.02	0.02

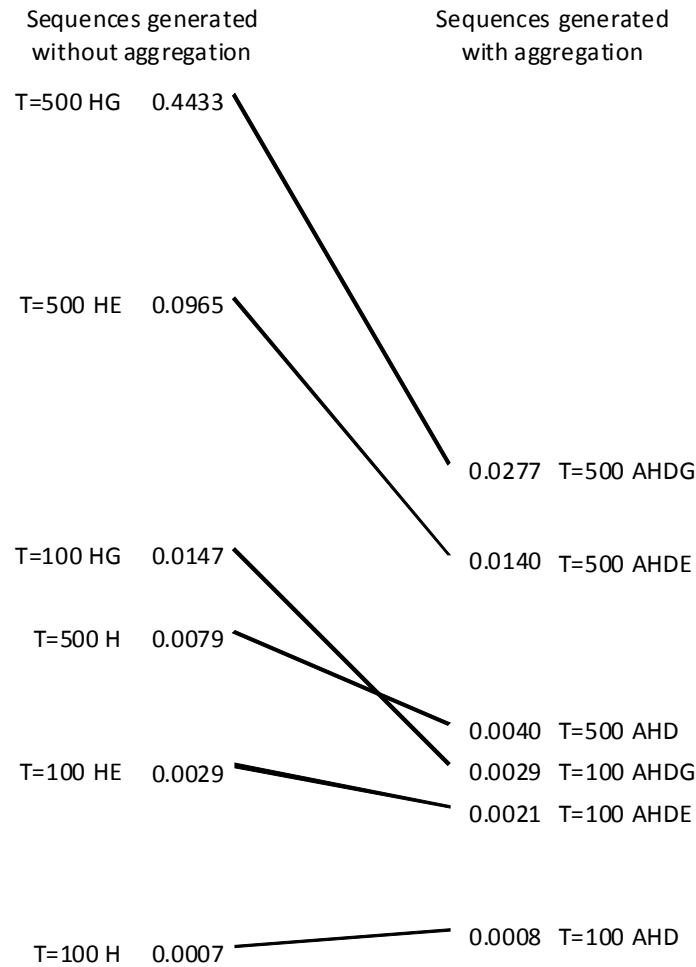


Figure 2. Average time (seconds) required to generate sequences.
Results are averaged over all instances with the given value of T .
Note that the vertical axis is logarithmic to improve readability.

To understand these results, consider a set of objects with the same count. The stride scheduling algorithm will put copies of these objects next to each other, which leads to poor-quality sequences, especially as the number of objects n approaches the number of positions T . Aggregation groups these objects, and the positions assigned to the new objects by stride scheduling are distributed across the aggregated sequence. The positions assigned to this set of objects in the disaggregated sequence are also distributed, leaving space for the objects with higher counts. As n approaches T , the sequences formed using aggregation are nearly ideal.

This explanation hints that aggregation would not perform as well if other heuristics were used. The results in [16, 18, 19, 20] show, however, that, when used with other heuristics, including those that perform better than stride scheduling, aggregation finds sequences that are more fair than those generated without using aggregation. In addition, aggregation is robust in the sense that changing the sequencing algorithm may make little difference in the quality of

the sequences generated. In this paper, we emphasize the results with stride scheduling because it has been used on all of these problems and in other work [7, 11, 13].

We also measured the clock time needed to generate these policies. Figure 2 shows the average time needed to generate the different solutions for different heuristics and different values of T . These are averages over all of the corresponding problem sets. As T increased, the time required increased for all heuristics.

Compared with the time required to generate the H sequences, using aggregation with stride scheduling to generate the AHD sequences required more time when $T = 100$ but required less time when $T = 500$. (Using aggregation also required less time when T was larger [19].) Reducing the number of objects by aggregation (from hundreds to around ten) reduces the computational effort of the stride scheduling algorithm. The extra effort of aggregating and disaggregating did not add much time for the larger instances.

Because the H sequences had very large values of RTV, generating the HE sequences (to reduce RTV) required a great deal of time, but improving the AHD sequences to generate the AHDE sequences required less additional time. Overall, generating the AHDE sequences required less time than generating the HE sequences.

Generating the HG solutions (to improve average waiting time) requires more time than generating the H solutions. The quality of the HG solutions is significantly better than the quality of the H solutions for some values of n , which shows that the extra effort generated some benefit. Although improving the AHD sequences to generate the AHDG sequences required additional time, generating the AHDG sequences required less time than generating the HG sequences.

Overall, these results show that using aggregation with the stride scheduling algorithm generates sequences that are more fair (measured with different metrics) with less computational effort. In particular, the sequences are more balanced (they have lower count-balances and gap-balances), they have smaller values for RTV, and they have less waiting time.

9 Summary and Conclusions

This paper presents an aggregation approach for generating fair sequences. We showed that using this approach with the well-known stride scheduling algorithm generates better solutions more quickly. This results holds for different measures of fairness, including count balance, gap balance, RTV, and average waiting time and for a range of problem sizes.

The aggregation approach presented here should be effective at generating good sequences for other fair sequencing problems in which the objective is to minimize the deviation from a perfectly fair sequence, including scheduling just-in-time manufacturing [9] and the bottleneck problem [13]. Problems in which one must satisfy hard constraints on the deviations (like the periodic maintenance scheduling problem and the Liu-Layland problem) will require a different approach, though aggregation could be used to accelerate the search for feasible solutions.

Acknowledgements This work was originally motivated by a collaboration with Leonard Taylor at the University of Maryland Medical Center.

References

1. Kubiak, W., Fair sequences, in Handbook of Scheduling: Algorithms, Models and Performance Analysis, Leung, J.Y-T., editor, Chapman & Hall/CRC, Boca Raton, Florida, 2004.

2. Bar-Noy, Amotz, Randeep Bhatia, Joseph Naor, and Baruch Schieber, Minimizing service an operation costs of periodic scheduling, *Mathematics of Operations Research*, 27, 518-544, 2002.
3. Bar-Noy, Amotz, Aviv Nisgah, and Boaz Patt-Shamir, Nearly Optimal Perfectly Periodic Schedules, *Distributed Computing*, 15, 207-220, 2002.
4. Wei, W.D., and Liu, C.L., On a Periodic Maintenance Problem, *Operations Research Letters*, 2, 90-93, 1983.
5. Park, Kyung S., and Doek K. Yun, Optimal Scheduling of Periodic Activities, *Operations Research*, 33, 690-695, 1985.
6. Campbell, Ann Melissa, and Jill R. Hardin, Vehicle Minimization for Periodic Deliveries, *European Journal of Operational Research*, 165, 668-684, 2005.
7. Waldspurger, C.A., and Weihl, W.E., Stride Scheduling: Deterministic Proportional-Share Resource Management, Technical Memorandum MIT/LCS/TM-528, MIT Laboratory for Computer Science, Cambridge, Massachusetts, 1995.
8. Balinski, M.L., and H.P. Young, Fair Representation, Yale University Press, New Haven, Connecticut, 1982.
9. Miltenburg, J., Level Schedules for Mixed-Model Assembly Lines in Just-in-Time Production Systems, *Management Science*, 35, 192-207, 1989.
10. Inman, R.R., and Bulfin, R.L., Sequencing JIT Mixed-Model Assembly Lines, *Management Science*, 37, 901-904, 1991.
11. Corominas, Albert, Wieslaw Kubiak, and Natalia Moreno Palli, Response Time Variability, *Journal of Scheduling*, 10, 97-110, 2007.
12. Garcia, A., R. Pastor, and A. Corominas, Solving the Response Time Variability Problem by Means of Metaheuristics, in *Artificial Intelligence Research and Development*, edited by Monique Polit, T. Talbert, and B. Lopez, pages 187-194, IOS Press, 2006.
13. Kubiak, Wieslaw, *Proportional Optimization and Fairness*, Springer, New York, 2009.
14. Rogers, David F., Robert D. Plante, Richard T. Wong, and James R. Evans, Aggregation and Disaggregation Techniques and Methodology in Optimization, *Operations Research*, 39, 553-582, 1991.
15. Herrmann, Jeffrey W., Generating Cyclic Fair Sequences using Aggregation and Stride Scheduling, Technical Report 2007-12, Institute for Systems Research, University of Maryland, College Park, 2007.
16. Herrmann, Jeffrey W., Using Aggregation to Reduce Response Time Variability in Cyclic Fair Sequences, to appear in *Journal of Scheduling*. DOI: 10.1007/s10951-009-0127-7. Published online on August 15, 2010.
17. Herrmann, Jeffrey W., Constructing Perfect Aggregations to Eliminate Response Time Variability in Cyclic Fair Sequences, Technical Report 2008-29, Institute for Systems Research, University of Maryland, October, 2008.
18. Herrmann, Jeffrey W., Generating Cyclic Fair Sequences for Multiple Servers, *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2009)*, Dublin, Ireland, August 10-12, 2009.
19. Herrmann, Jeffrey W., Using Aggregation to Construct Periodic Policies for Routing Jobs to Parallel Servers with Deterministic Service Times, to appear in *Journal of Scheduling*. DOI: 10.1007/s10951-010-0209-6. Published online on November 24, 2010.
20. Herrmann, Jeffrey W., Aggregating Alphabets to Construct Balanced Words, Technical Report 2009-12, Institute for Systems Research, University of Maryland, September, 2009.
21. Sano, Shinya, Naoto Miyoshi, and Ryohei Kataoka, m balanced words: A generalization of balanced words, *Theoretical Computer Science*, Volume 314, Issues 1-2, 25 February 2004, pages 97-120, 2004.
22. van der Laan, D.A., Routing jobs to servers with deterministic service times, *Mathematics of Operations Research*, 30(1), 195-224, 2005.

Minimizing Value-at-Risk in the Single-Machine Total Weighted Tardiness Problem

Semih Atakan · Birce Tezel · Kerem
Bülbül · Nilay Noyan

Abstract The vast majority of the machine scheduling literature focuses on deterministic problems, in which all data is known with certainty a priori. This may be a reasonable assumption when the variability in the problem parameters is low. However, as variability in the parameters increases incorporating this uncertainty explicitly into a scheduling model is essential to mitigate the resulting adverse effects. In this paper, we consider the celebrated single-machine total weighted tardiness (TWT) problem in the presence of uncertain problem parameters. We impose a probabilistic constraint on the random TWT and introduce a risk-averse stochastic programming model. In particular, the objective of the proposed model is to find a non-preemptive static job processing sequence that minimizes the value-at-risk (VaR) measure on the random TWT at a specified confidence level. Furthermore, we develop a lower bound on the optimal VaR that may also benefit alternate solution approaches in the future. In this study, we implement a tabu-search heuristic to obtain reasonably good feasible solutions and present results to demonstrate the effect of the risk parameter and the value of the proposed model with respect to a corresponding risk-neutral approach.

Keywords single-machine; weighted tardiness; stochastic processing times; stochastic scheduling; value-at-risk; probabilistic constraint; stochastic programming.

Semih Atakan
Manufacturing Systems and Industrial Engineering, Sabancı University, Istanbul, Turkey
E-mail: semihatakan@sabanciuniv.edu

Birce Tezel
Manufacturing Systems and Industrial Engineering, Sabancı University, Istanbul, Turkey
E-mail: btezel@sabanciuniv.edu

Kerem Bülbül
Manufacturing Systems and Industrial Engineering, Sabancı University, Istanbul, Turkey
E-mail: bulbul@sabanciuniv.edu

Nilay Noyan
Manufacturing Systems and Industrial Engineering, Sabancı University, Istanbul, Turkey
E-mail: nnoyan@sabanciuniv.edu

1 Introduction

The weighted tardiness objective is a classical due date related performance measure in make-to-order environments. The goal is to find a job (order) processing sequence in order to minimize the total cost incurred due to missed due dates. For a given job, the cost is directly proportional to the associated tardiness. The unit tardiness cost (weight) may either be associated with the perceived penalty due to a loss of customer goodwill or may represent actual contractual penalties. The interested reader is referred to Sen et al (2003) for a relatively recent survey on the topic.

In the traditional single-machine TWT problem described above, all processing times, release dates, due dates, and weights are known in advance at time zero with certainty when the job sequencing decision is taken. However, in many practical settings the exact values of one or several of these parameter types may not be available at the time the dispatcher determines a job processing sequence. In particular, possible machine breakdowns, variable sequence-independent setup times, inconsistency of the worker performance, or changes in tool quality may introduce uncertainty into the processing times. The uncertainty in the processing time of a job is resolved at the time of the job completion. The models developed in this paper are sufficiently general to incorporate randomness into all parameters. However, from a practical point of view it is reasonable to presume that a due date is quoted as a result of a mutual agreement with the customer, and the unit tardiness weight associated with a customer is also known based on either the internal priority of the customer or the contractual agreement. Therefore, in our computational experiments the due dates and the unit tardiness weights are deterministic. Furthermore, we assume that all jobs are ready to be released at time zero. Consequently, we focus on the uncertainty in the processing times which leads to uncertain completion times and tardiness values. Our objective is to determine a risk-averse fixed job processing sequence at time zero that hedges against the uncertainty in the processing times. In the stochastic scheduling terminology (see Pinedo (2008)), we construct a non-preemptive static list policy.

Traditional models for decision making under uncertainty define optimality criteria based on expected values and disregard variability inherent in the system. Following this mainstream risk-neutral approach, most of the classical stochastic scheduling puts a lot of effort into analyzing the expected performance by assuming that uncertain parameters such as processing times follow specific distributions. See Pinedo (2008) for an excellent overview of conventional stochastic scheduling. However, variability typically implies a deterioration in performance, and risk-neutral models may provide solutions that perform poorly under certain realizations of the random data. Capturing the effect of variability can be accomplished by incorporating the appropriate risk measures into the model that reflect the preferences of the decision maker. Several criteria to select risk measures have been discussed in the literature (see, e.g., Ogryczak and Ruszczyński (1999, 2002); Artzner et al (1999)). Considering the wide range of criteria, there is no universally accepted single risk measure appropriate for all decision making contexts. In this study, we consider the VaR measure which is a very popular and widely applied risk measure in the finance literature. For the studies related to VaR we refer to the chapter by Larsen et al (2002). In our context, we focus on the TWT as the random outcome associated with a fixed job processing sequence selected at time zero. The goal is to specify the smallest possible upper bound on the random TWT that will be exceeded with at most a pre-specified small probability. Here, the selected upper bound is the VaR of the random TWT at the desired probability level, and we minimize

VaR. The concept of VaR is closely related to probabilistic constraints. Stochastic programming models with probabilistic constraints were introduced by Charnes et al (1958) and have been employed successfully in a variety of fields. The interested reader is referred to Prékopa (1995) and Dentcheva (2006) for reviews and a comprehensive list of references. Our proposed approach is an intuitive and practical way of modeling a service level requirement for the TWT under the stochastic setup and leads to a novel risk-averse stochastic programming model. To the best of our knowledge, this is a first in the machine scheduling literature.

It is well known that models incorporating VaR exhibit a non-convex structure even if the underlying deterministic problem is convex. The existing solution methods primarily deal with VaR integrated into a linear program (LP). Thus, the decision variables are continuous, and VaR is introduced on a random outcome expressed as a linear function of the decision variables. Larsen et al (2002) provides a review of the algorithms available for solving such problems. Note that these studies are generally concerned with portfolio optimization problems. Larsen et al (2002) also introduces two heuristic algorithms which solve a series of problems involving a related risk measure known as conditional-value-at-risk (CVaR). In contrast to VaR, the problem of minimizing CVaR can be formulated as an LP if the uncertainty is represented by a set of scenarios, and the proposed heuristics use LP techniques iteratively. However, in our study the underlying problem involves sequencing decisions that can only be expressed by employing binary variables; and therefore, even minimizing CVaR is hard. Consequently, the existing solution methods do not apply in our case.

We characterize the randomness associated with the uncertain parameters by a finite set of scenarios, where a scenario represents a joint realization of all random parameters. It is important to point out that the scenario approach allows us to generate data from any distribution and to model, for instance, the correlation of the random processing times among different jobs by considering their joint realizations. In this sense, a scenario-based approach is more general than assuming specific distributions. On the down side, the computational complexity of solving the problem is closely affected by the number of scenarios. There are only a few studies utilizing a scenario-based approach for machine scheduling problems. For example, Gutjahr et al (1999) minimize the expected TWT with stochastic processing times and propose a stochastic branch-and-bound technique, where a sampling approach is embedded into their bounding schemes. Alternatively, other existing scenario-based studies develop robust optimization models in order to optimize the worst-case performance over all scenarios. Such a worst-case analysis does not require the probabilities of the scenarios. The sum of completion times is employed in Daniels and Kouvelis (1995); Yang and Yu (2002), and the weighted sum of completion times is considered by de Farias et al (2010), while Kasperski (2005) focuses on the maximum lateness as the random performance criterion. One or several of the robustness measures known as the maximum deviation from optimality, the maximum relative deviation from optimality, and the maximum value over all scenarios are incorporated in these papers. Except de Farias et al (2010), all these studies design specialized algorithms for the robustness measure and random performance criterion of interest. de Farias et al (2010) identify a family of valid inequalities to strengthen the mixed-integer formulation of their problem. Furthermore, Alouloua and Croce (2008) provide several complexity results in the domain of robust scheduling. In contrast to robust approaches adopting a conservative worst-case view, we define our optimality criterion based on VaR which is a quantile of the random outcome at a specified probability level. That is, we utilize probabilis-

tic information and develop a risk-averse stochastic programming model alternative to existing robust optimization models. Note that setting the required probability level to one subsumes the robust optimization problem of minimizing the maximum TWT over all scenarios. However, when the required probability level is specified as $\alpha < 1$, we minimize the maximum TWT over a subset of scenarios with an aggregate probability of at least α . Our risk-averse model identifies the optimal subset of scenarios with the specified minimum aggregate probability level and minimizes the maximum TWT over this subset. Thus, it is less conservative than the robustness approach which considers all scenarios.

The main contribution of this study is to develop a risk-averse model that is novel in machine scheduling. We analyze the behavior of the proposed model in comparison to that of the risk-neutral model and provide insights on the impact of the risk preference. Furthermore, in all papers on robust scheduling mentioned above the corresponding deterministic single-machine problems are polynomially solvable. However, the single-machine TWT problem is strongly $\mathcal{NP}$ -hard (Lenstra et al (1977)), and incorporating VaR poses additional computational difficulties. Thus, we also implement a tabu search algorithm to solve the proposed model heuristically.

In the next section, we formally define the risk-averse TWT problem and present a mathematical programming formulation. In Section 3, we introduce a lower bounding scheme for our problem and also briefly discuss the implementation details of the tabu search. Computational results are presented in Section 4, and we conclude in Section 5 with further research directions.

2 Stochastic Programming Model

In this section, we first present the underlying deterministic model of the stochastic TWT problem. Then, we discuss how to model the uncertainty inherent in the system and develop our risk-averse stochastic programming model.

2.1 Underlying deterministic model

For single-machine scheduling problems, four frequently used alternate deterministic formulations appear in the literature (see Keha et al (2009)): disjunctive (DF), time-indexed (TIF), linear ordering (LOF), and the assignment and positional date formulations (APDF). TIF has a tight LP relaxation and is the best contender among these four formulations if the processing times are small. TIF, however, cannot be adapted to our stochastic setting directly, because it infers the sequence from the completion times represented by binary decision variables. Recall that our goal is to find a non-preemptive static job processing sequence at time zero. That is, the decisions are independent of the random realizations of data, and therefore, relying on completion time information that is contingent on the random processing times (and random release dates if applicable) is not appropriate to construct a static job processing sequence. Our preliminary results indicate that DF is outperformed by LOF and APDF. This observation is also supported by the extensive computational study presented in Keha et al (2009). Thus, among the common formulations only LOF and APDF are viable options for our proposed risk-averse model. In this study, we work with LOF but note that the proposed modeling framework would apply to APDF in a similar way.

We define the set of jobs to be processed as $N := \{1, \dots, n\}$, where n denotes the number of jobs. Associated with each job $j \in N$ are several parameters: a processing time p_j , a due date d_j , and a tardiness cost per unit time w_j if job j completes processing after d_j . The completion time of job j is represented by C_j , and the tardiness T_j of job j is then expressed by $T_j = \max(0, C_j - d_j)$. The binary variable δ_{jk} takes the value 1, if job j precedes job k in the processing sequence, and is zero otherwise. By convention, we set $\delta_{jj} = 1$ for all $j \in N$. Assuming zero release dates, the deterministic single-machine TWT problem, described as $1/\sum_j w_j T_j$ following the common three field notation of Graham et al (1979), is formulated below:

$$\min \sum_{j=1}^n w_j T_j$$

subject to

$$\delta_{jk} + \delta_{kj} = 1, \quad 1 \leq j < k \leq n, \quad (1)$$

$$\delta_{jk} + \delta_{kl} + \delta_{lj} \leq 2, \quad \forall j, k, l \in N : j \neq k, k \neq l, l \neq j, \quad (2)$$

$$C_j = \sum_{k \in N} p_k \delta_{kj}, \quad \forall j \in N, \quad (3)$$

$$T_j \geq C_j - d_j, \quad \forall j \in N, \quad (4)$$

$$T_j \geq 0, \quad \forall j \in N, \quad (5)$$

$$\delta_{jk} \in \{0, 1\}, \quad \forall j, k \in N. \quad (6)$$

Constraints (1) ensure that for each pair of jobs j and k either job j precedes job k or vice versa. Constraints (2) represent the transitivity requirements for a linear ordering of the jobs. In other words, they guarantee that for any triplet of jobs j, k, l , if job j precedes job k and job k precedes job l then job j precedes job l . The completion time C_j of job j is the sum of the processing times of all of its predecessors (recall that $\delta_{jj} = 1$ by convention) as prescribed by (3), and T_j is related to C_j by constraints (4) and (5). Constraints (6) are the binary variable restrictions required for the sequencing decisions.

In our setting, the actual values of the processing times are not certain at the time we determine the job processing sequence, and the processing times can be represented by random variables. This implies that the completion times and the tardiness values associated with a sequence are also random variables, since they are functions of the random processing times. In this case, comparing alternate candidate sequences requires comparing their respective random TWT values. In this paper, we propose a risk-averse approach which evaluates a sequence with respect to a certain quantile of the distribution of the associated random TWT. Let Υ and ξ_j denote the random TWT and the random processing time of job $j \in N$, respectively. The random variable Υ is a random outcome associated with a sequence $\delta \in \{0, 1\}^{n \times n}$. We can represent it as a function of the decision vector $\delta \in \{0, 1\}^{n \times n}$ as below:

$$\Upsilon = \sum_{j=1}^n w_j \max \left(\sum_{k=1}^n \xi_k \delta_{kj} - d_j, 0 \right). \quad (7)$$

We intend to model the risk associated with the variability of the random outcome Υ by introducing the following probabilistic constraint:

$$P(\Upsilon \leq \theta) \geq \alpha, \quad (8)$$

where α is a specified large probability such as 0.90 or 0.95. Here θ denotes an upper bound on the TWT that is exceeded with at most a small probability of $1 - \alpha$. If $\alpha = 1$, $\Upsilon \leq \theta$ holds almost surely. As discussed in more depth in Section 1, such a probabilistic constraint is intuitive and allows us to model a service level requirement for the TWT under the stochastic setup. We refer to α as the risk parameter which reflects the level of risk-aversion of the decision maker. Clearly, increasing α results in allowing a higher value of the upper bound θ . We propose not to specify the value of θ as an input, but consider it as a decision variable with the purpose of identifying the sequence with the smallest possible value of θ given the risk aversion of the decision maker. Thus, in our model we minimize θ for a specified parameter α , which is equivalent to minimizing the α -quantile of the random TWT. The α -quantile has a special name in risk theory as presented in the next definition.

Definition 1 Let X be a random variable. The α -quantile

$$\inf\{\eta \in \mathbb{R} : F_X(\eta) \geq \alpha\}$$

is called the Value at Risk (VaR) at the confidence level α and denoted by $\text{VaR}_\alpha(X)$, $\alpha \in (0, 1]$.

The probabilistic constraint (8) can equivalently be formulated as a constraint on the VaR of the random TWT:

$$\text{VaR}_\alpha(\Upsilon) \leq \theta. \quad (9)$$

In other words, by considering the proposed probabilistic constraint (8) we specify the VaR as the risk measure on the random TWT, and minimizing θ corresponds to seeking the sequence with the smallest possible VaR value for a specified α value.

A model with a probabilistic constraint similar to that in (8) with randomness on the left hand side was first studied by de Panne and Popp (1963) and Kataoka (1963). Kataoka introduces a transportation type model and Van de Panne and Popp present a diet (cattle feed) optimization model with a single probabilistic constraint. In these studies, the random outcome of interest is a linear function of the decision vector, and in both studies the solution methods are specific to random coefficients with a joint normal distribution. In contrast, the random outcome Υ in our work is not a linear function of the decision vector as evident from (7), and we do not assume that it has a special distribution.

We characterize the random processing times by a finite set of scenarios denoted by S , where a scenario represents a joint realization of the processing times of all jobs. To develop our stochastic programming formulation, previously introduced parameters and variables are augmented with scenario indices:

p_j^s : processing time of job j under scenario s , $s \in S$.

T_j^s : tardiness of job j under scenario s , $s \in S$.

C_j^s : completion time of job j under scenario s , $s \in S$.

π^s : probability of scenario s , $s \in S$.

Then, we formulate the problem of minimizing the VaR in the single-machine TWT problem as follows:

$$\min \quad \theta \quad (10)$$

$$\text{subject to} \quad \delta_{jk} + \delta_{kj} = 1, \quad 1 \leq j < k \leq n, \quad (11)$$

$$\delta_{jk} + \delta_{kl} + \delta_{lj} \leq 2, \quad \forall j, k, l \in N : j \neq k, k \neq l, l \neq j, \quad (12)$$

$$C_j^s = \sum_{k \in N} p_k^s \delta_{kj}, \quad \forall j \in N, s \in S, \quad (13)$$

$$T_j^s \geq C_j^s - d_j, \quad \forall j \in N, s \in S, \quad (14)$$

$$T_j^s \geq 0, \quad \forall j \in N, s \in S, \quad (15)$$

$$\sum_{j \in N} w_j T_j^s - \theta \leq T_{\max}^s \beta^s, \quad \forall s \in S, \quad (16)$$

$$\sum_{s \in S} \pi^s \beta^s \leq 1 - \alpha, \quad (17)$$

$$\beta^s \in \{0, 1\}, \quad \forall s \in S, \quad (18)$$

$$\delta_{jk} \in \{0, 1\}, \quad \forall j, k \in N. \quad (19)$$

We emphasize that the constraints (11), (12), and (19) in the model above are identical to the constraints (1), (2), and (6) in the deterministic model, respectively. That is, the sequencing decisions are independent of the uncertainty. Constraints (13)-(15) are the counterparts of the constraints (3)-(5) in the deterministic model, respectively, and calculate the resulting job completion times under each joint realization of the processing times. The parameter $T_{\max}^s$ is a scenario-specific sufficiently large number which guarantees that the binary variable β^s is set to 1 by the corresponding constraint (16) if the realization of the TWT under scenario s exceeds the threshold value θ . Constraint (17) mandates that the probability of exceeding the threshold value θ for the random TWT is no more than $1 - \alpha$. At an optimal solution of the formulation above, T_j^s may be strictly larger than $\max\{C_j^s - d_j, 0\}$ for some scenario $s \in S$ because the tardiness values are not associated with positive cost coefficients in the objective. Obviously, we preserve optimality by setting $T_j^s = \max\{C_j^s - d_j, 0\}$. For the validity of the formulation (10)-(19), we must also ensure that $T_{\max}^s$ is no smaller than the maximum possible TWT under scenario s . In order to obtain a reasonably tight formulation, we sort the processing times under scenario s in non-increasing order and denote the j th largest processing time under scenario s by $p_{[j]}^s$. Then, the maximum possible completion time of the k th job in the sequence, $k = 1, \dots, n$, under scenario s is computed as $C_{[k]}^s = \sum_{j=1}^k p_{[j]}^s$. Next, the due dates and the unit tardiness weights are assigned to the completion times in non-increasing and non-decreasing order, respectively. A standard pairwise interchange argument (not necessarily adjacent) demonstrates that the resulting TWT is an upper bound on the TWT of any job processing sequence under scenario s .

Uncertainty in the due dates and/or the unit tardiness weights may be incorporated in our formulation in a straightforward manner by replacing the parameters d_j and w_j by d_j^s and w_j^s in the constraints (14) and (16), respectively. This modification does not affect the number of variables and constraints. However, if the release dates are not known in advance, then the completion time constraints (13) must be replaced by the following set of constraints adapted from the deterministic formulation in Nemhauser and Savelsbergh (1992):

$$C_j^s \geq r_i^s \delta_{ij} + \sum_{\{k : r_k^s < r_i^s, k \neq j\}} p_k^s (\delta_{ik} + \delta_{kj} - 1) + \sum_{\{k : r_k^s \geq r_i^s\}} p_k^s \delta_{kj}, \quad \forall i, j \in N.$$

In the remainder of the paper, we refer to the formulation (10)-(19) as VaR-TWT.

3 On the Solution Method

The main intent of our work is to argue and illustrate the value of the risk-averse single-machine TWT problem. Therefore, our efforts are focused primarily on modeling and the insights provided by the computational results. Clearly, developing an effective solution method to solve VaR-TWT is an ambitious endeavor that we will delve into in the future. In this larger scheme, we first develop a lower bound on the optimal VaR that may benefit prospective studies as well. This lower bounding mechanism is employed for computing the optimality gaps in our computational study in Section 4 when it is too expensive to solve VaR-TWT to optimality. Then, we briefly discuss some implementation details of a standard tabu search heuristic that yields reasonably high-quality feasible solutions to the risk-averse single-machine TWT problem.

3.1 Lower Bounding Scheme

The relation of stochastic dominance is one of the fundamental concepts to compare random variables (Mann and Whitney (1947); Lehmann (1955)). It introduces a pre-order in the space of real random variables. We refer to Muller and Stoyan (2002) for a detailed and comprehensive discussion on stochastic dominance relations. In a stochastic dominance based approach, random variables are compared by a point-wise comparison of some performance functions constructed from their distribution functions. In this study, we utilize the first-order stochastic dominance (FSD) which considers the cumulative distribution function itself as the performance function. Let F_X and F_Y denote the distribution functions of the random variables X and Y , respectively. The FSD relation between X and Y is defined as below:

Definition 2 A random variable X dominates another random variable Y in the first order; that is, X is stochastically larger than Y , if

$$F_X(\eta) \leq F_Y(\eta) \quad \text{for all } \eta \in \mathbb{R}. \quad (20)$$

This ordering is denoted by $X \succeq_{(1)} Y$.

It is easy to see that by the definition of the FSD relation we have

$$\left[X \succeq_{(1)} Y \right] \Leftrightarrow \left[\text{VaR}_\alpha(X) \geq \text{VaR}_\alpha(Y) \quad \text{for all } 0 < \alpha \leq 1 \right]. \quad (21)$$

We leverage on this fundamental relation between the concepts of VaR and FSD in order to obtain a lower bound on the optimal objective value of VaR-TWT. We consider a finite probability space where the sample space is given by $\Omega = \{\omega_1, \dots, \omega_N\}$ with corresponding probabilities $p_1, \dots, p_N$. Let $y_i = Y(\omega_i)$, $i = 1, \dots, N$, and $x_i = X(\omega_i)$, $i = 1, \dots, N$, denote the realizations of the random variables Y and X , respectively. In our study, we are interested in the random TWT. In particular, the realizations of the random variable Y are obtained by solving a single-machine TWT problem independently for each scenario. On the other hand, the random variable X denotes the random TWT associated with the optimal sequence of the problem VaR-TWT. Next, we state formally that $\text{VaR}_\alpha(Y)$ is a lower bound on the optimal VaR obtained by solving VaR-TWT for any given fixed α .

Proposition 1 *Let Y represent a random variable, where the realization $Y(\omega_i)$ is equal to the TWT associated with the sequence that minimizes the TWT under scenario i , $i = 1, \dots, N$. Furthermore, the random variable X denotes the random TWT associated with the optimal sequence δ^* of the problem VaR-TWT. Then, $\text{VaR}_\alpha(Y) \leq \text{VaR}_\alpha(X)$ for all $0 < \alpha \leq 1$.*

Proof $X(\omega_i)$ is the TWT associated with the sequence δ^* under scenario i . Since δ^* is a feasible sequence for the problem of minimizing the TWT under scenario i , we have $X(\omega_i) \geq Y(\omega_i)$ for all $i = 1, \dots, N$. It trivially follows that $P(X \leq \eta) \leq P(Y \leq \eta)$ for all $\eta \in R$, i.e., X dominates Y in the first-order. Consequently, $\text{VaR}_\alpha(Y) \leq \text{VaR}_\alpha(X)$ for all $0 < \alpha \leq 1$ by (21).

Note that the random variable Y does not have a special interpretation in the context of our problem. It only serves the purpose of obtaining a valid lower bound on the optimal objective function value of our problem.

Calculating the lower bound in Proposition 1 is strongly $\mathcal{NP}$ -hard because it requires solving $|S|$ instances of the deterministic single-machine TWT problem. A remedy to this issue is constructing a lower bound on the optimal TWT under each scenario. Then, the realizations $Y(\omega_i)$, $i = 1, \dots, N$, in Proposition 1 are replaced with these lower bounds. The interested reader is referred to Tanaka et al (2009); Pan and Shi (2007); Sen (2010) for a variety of lower bounds for the deterministic single-machine TWT problem.

3.2 Tabu Search

For many combinatorial optimization problems in scheduling and routing, local search algorithms, and in particular metaheuristics, provide us with established and viable solution methods. Given the additional complexity stemming from the consideration of risk in the single-machine TWT problem, it is of interest to gauge the potential of local search algorithms for this problem. Thus, in this study we implemented a basic tabu search algorithm for a preliminary assessment. A further important consideration is whether this tabu search algorithm fares well compared to a risk-neutral optimization model to be introduced in Section 4. In other words, we intend to figure out whether we can do at least as good as an expectation-based model even with a simple metaheuristic. In the rest of this section we briefly discuss some details of the tabu search implementation.

The feasible region of the risk-averse single-machine TWT problem is identical to that of its deterministic counterpart with $n!$ feasible sequences. Therefore, it suffices to conduct a search over the job processing sequence. In the deterministic version, a well-known dominance rule (see Pinedo (2008)) mandates that there exists an optimal sequence in which job i precedes job j if $d_i \leq d_j$, $p_i \leq p_j$, and $w_i \geq w_j$. We adapt this rule to our problem by replacing the processing times by their expected values, where the expected processing time of job j is given by $\bar{p}_j = \sum_{s \in S} \pi^s p_j^s$. We determine the priority of job j as $\frac{w_j}{\bar{p}_j \times d_j}$ and pick the jobs in non-increasing order of their priorities to construct an initial feasible sequence. Clearly, in the future we need to conceive and incorporate dominance rules that are more tailored toward the risk-averse nature of our problem.

The neighborhood generator is adjacent pairwise interchange. Note that function evaluations are expensive for our problem. Even for this basic neighborhood, the randomness in the processing times (and possibly in the other parameters of the problem) does not allow to calculate the objective value of a neighboring solution in a straightforward way based on the objective value of the current solution. Therefore, the small size of this neighborhood makes it attractive from a computational speed point of view. Features such as approximate move evaluations may need to be incorporated for an effective local search heuristic for larger instances.

The search is guided with a best-improve strategy. That is, at each iteration we pick the best of the $n - 1$ neighboring solutions. The tabu list keeps track of the jobs not to be interchanged for a number of iterations. The size of the tabu list is limited to $\lceil 5n/2 \rceil$. A tabu move is accepted only if the associated objective value exceeds the best solution found so far. We terminate the algorithm at $500n$ iterations or if the best objective value does not improve for $15n$ iterations, or if all moves in the current neighborhood are tabu.

4 Computational Study

The goals of our computational study are two-fold. In the first part, we demonstrate that the lower bounding procedure developed in Section 3.1 provides good bounds in this preliminary computational study. Furthermore, the results indicate that the basic tabu search algorithm of Section 3.2 yields feasible solutions of reasonable quality for the risk-averse single-machine TWT problem for instances with up to 20 jobs. In the second part, the value of the proposed risk-averse model is investigated with respect to that of a risk-neutral model.

All runs were conducted on a single core of an HP Linux workstation with two Intel®Xeon®W5580 3.20GHz CPU's and 32 GB of memory. The integer programming formulations were solved by CPLEX 11.2, and the tabu search was implemented in C++.

4.1 Generation of problem instances

While our modeling framework allows for randomness in all problem parameters, we focus on the uncertainty in the processing times in our computational study as justified by the discussion in Section 1. For each instance, we generate a set of equally likely scenarios representing the joint realizations of the processing times by adding negative or positive perturbations to each estimated processing time $\hat{p}_j$, where $\hat{p}_j$ follows an integer uniform distribution $U[1, 100]$ for $j = 1, \dots, n$. To this end, let ε_j denote the random perturbation for job j , where ε_j^s is the realization of ε_j for scenario s . Then, the processing time of job j under scenario s is given by $p_j^s = \hat{p}_j + \varepsilon_j^s$. In our first set of experiments, we set $\varepsilon_j \sim U(-\hat{p}_j/4, \hat{p}_j/3)$, which results in $E(\hat{p}_j + \varepsilon_j) = \hat{p}_j + \hat{p}_j/24$ and a coefficient of variation (CV) of 0.16. CV is a normalized measure of dispersion and is defined as $CV(\hat{p}_j + \varepsilon_j) = \text{standard deviation}(\hat{p}_j + \varepsilon_j) / E(\hat{p}_j + \varepsilon_j)$ for the processing time of job j . Additional data with different CV values are generated to further analyze the value of the risk-averse model in Section 4.3. The details are presented in Table 2.

In the literature, it is well established that the tightness and the range of the due dates is a primary determinant of difficulty for due date related problems. Thus, by following the popular scheme of Potts and van Wassenhove (1982), we first generate

the due dates from a discrete uniform distribution $[(1 - TF - RDD/2) \times \bar{P}], [(1 - TF + RDD/2) \times \bar{P}]$, where $\bar{P}$ is the sum of the expected processing times, i.e., $\bar{P} = \sum_{j=1}^n \sum_{s \in S} \pi_j^s p_j^s$. The tardiness factor TF is a rough estimate of the proportion of jobs that might be expected to be tardy in an arbitrary sequence (Srinivasan (1971)) and is set to 0.3. Hard instances generally result from such small values of TF (see Bulbul et al (2007); Sen (2010)). The due date range factor RDD is set to 0.2 to increase the contention around the mean due date. The weights are drawn from an integer uniform distribution $U[1, 100]$.

4.2 Computational performance of the Lower Bound and the Tabu Search

In the first part of our study, we generate 5 instances for each combination of $n = 10, 20, 30$, and $|S| = 100, 150, 200, 250, 300$, as described in the previous section. The risk parameter $\alpha = 0.90$. For each instance, we calculate the lower bound presented in Section 3.1 and then solve it by both VaR-TWT and the tabu search implemented. The results averaged over 5 instances appear in Table 1.

Table 1 Effectiveness of the tabu search and the lower bound ($\alpha = 0.90$).

		S				
		100	150	200	250	300
n = 10	UB on Gap - Tabu	1.93%	0.00%	3.95%	1.29%	1.02%
	UB on Gap - VaR-TWT	0.00%	0.00%	0.00%	0.37%	1.02%
	CPU - Tabu	0.16	0.24	0.34	0.41	0.48
	CPU - VaR-TWT	439.54	446.44	1263.48	5231.41	5195.23
n = 20	UB on Gap - Tabu	6.88%	7.27%	10.55%	1.80%	4.63%
	UB on Gap - VaR-TWT	0.80%	2.50%	3.13%	2.76%	4.05%
	CPU - Tabu	1.29	1.94	2.40	3.11	3.60
	CPU - VaR-TWT	7194.83	7195.69	7195.35	7197.25	7197.70
n = 30	UB on Gap - Tabu	5.91%	25.25%	14.46%	12.47%	8.20%
	UB on Gap - VaR-TWT	3.71%	6.45%	6.76%	16.20%	16.80%
	CPU - Tabu	4.39	6.49	9.04	10.07	12.17
	CPU - VaR-TWT	7199.31	7199.43	7199.34	7199.53	7198.64

The time limit for VaR-TWT is set to 7200 seconds, and if optimality is not proven in the time allotted, then we record both the best lower bound and the incumbent solution available from CPLEX. For a given instance and an algorithm, the optimality gap is computed with respect to the optimal solution if it is available. Otherwise, the best known lower bound is determined by taking the maximum of our lower bound and the best lower bound retrieved from CPLEX, and the optimality gap is computed with respect to this lower bound. For each n , the first two rows specify the average upper bounds on the optimality gaps (“UB on Gap - Tabu” and “UB on Gap - VaR-TWT”), and the associated average CPU times in seconds are reported in the last two rows.

Several conclusions may be drawn from Table 1. First, solving VaR-TWT is very time consuming even for small n as the number of scenarios grows. Only half of the instances with $n = 10$ and $|S| = 250, 300$, and none of instances with $n = 20, 30$ can be solved to optimality within the time limit. Second, our lower bounding scheme is quite tight for this data set. Note that for $n = 20, 30$, CPLEX returns a trivial lower

bound of 0 for all instances. That is, all optimality gaps for instances with 20 and 30 jobs are computed with respect to our lower bound. Furthermore, note that a standard time-indexed formulation is applied to the single-machine TWT problems to be solved for our lower bounding scheme, and if optimality is not achieved in 1000 seconds, then we use the best available lower bound instead of the optimal solution for calculating the lower bound based on Proposition 1. Third, the quality of solutions obtained from the tabu search is relatively high for $n = 10, 20$. However, the performance degrades considerably for $n = 30$.

4.3 Value of the Risk-Averse Model

The value of a risk-averse solution depends on the relative performance of the corresponding deterministic and risk-neutral solutions as a function of the risk appetite. Therefore, in this part, we benchmark VaR-TWT against corresponding deterministic and risk-averse models as the risk parameter α is varied. The deterministic counterpart of our problem is the conventional single-machine TWT problem, in which all processing times take on their expected values; that is, we have $p_j = \bar{p}_j = \sum_{s \in S} \pi^s p_j^s$. In the risk-neutral version of our problem, we minimize the expected TWT by solving the following formulation:

$$\min \sum_{j=1}^n w_j \sum_{s \in S} \pi^s T_j^s \quad (22)$$

$$\text{subject to} \quad (11) - (15) \text{ and } (19). \quad (23)$$

In Figure 1, we zoom into two instances from Table 1 to illustrate how the VaR changes as α is varied. For this data set we obtain risk-averse solutions without sacrificing much from the expected TWT as α increases. Finally, we create data with different variability in the processing times. To this end, we consider the uniform distribution and a mixture of two exponential distributions to generate perturbations. We specify the parameters of these distributions such that the resulting processing times have different CV values. The details are summarized in Table 2. All scenarios are assumed to be equally likely. A total of 10 instances for $n = 10, 20$ are solved by the risk-neutral model and tabu search for $\alpha = 0.90$. For these 10 instances, the entries in Table 3 indicate the relative decrease in VaR and the relative increase in the expected TWT for the solution of the tabu search in comparison to that of the risk-neutral model. The risk-averse solution exhibits significant improvements over the risk-neutral solution, albeit at times at the expense of the expected TWT to hedge against the uncertainty.

Table 2 Summary of parameters for creating processing times.

	Random Perturbation	Resulting Processing Times	
		$E(\hat{p}_j + \varepsilon_j)$	$CV(\hat{p}_j + \varepsilon_j)$
DataSet 1	$\varepsilon_j \sim U(-\hat{p}_j/4, \hat{p}_j)$	$\hat{p}_j + 3\hat{p}_j/8$	0.26
DataSet 2	$\varepsilon_j \sim U(-\hat{p}_j/4, 2\hat{p}_j)$	$\hat{p}_j + 7\hat{p}_j/8$	0.35
DataSet 3	$\varepsilon_j \sim \begin{cases} \exp(13\hat{p}_j/24) & \text{with prob of 0.75} \\ \exp(-\hat{p}_j/8) & \text{with prob of 0.25} \end{cases}$	$\hat{p}_j + 3\hat{p}_j/8$	0.40
DataSet 4	$\varepsilon_j \sim \begin{cases} \exp(6\hat{p}_j/5) & \text{with prob of 0.75} \\ \exp(-\hat{p}_j/10) & \text{with prob of 0.25} \end{cases}$	$\hat{p}_j + 7\hat{p}_j/8$	0.63

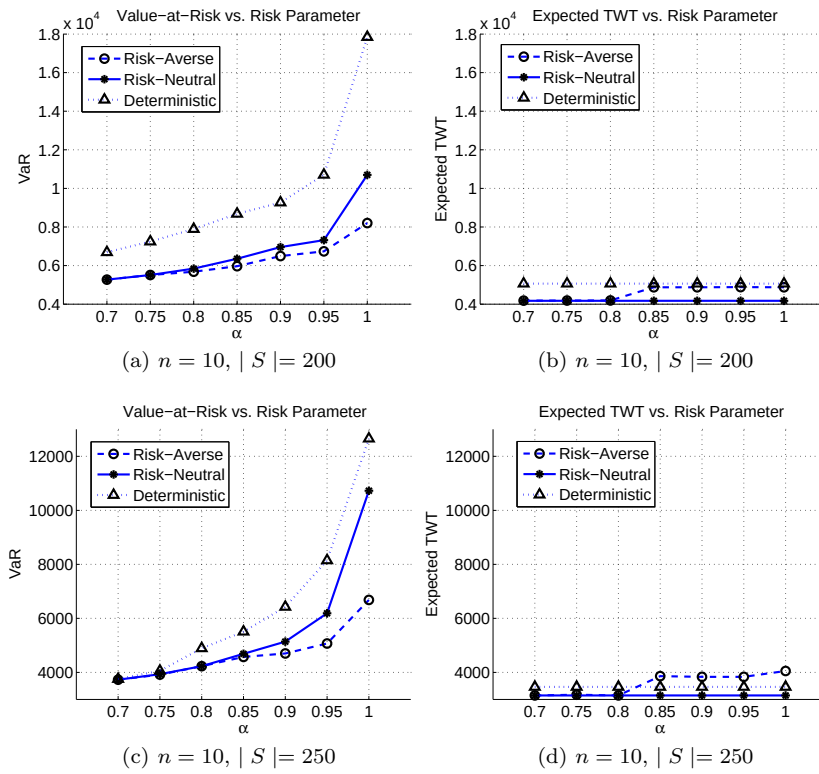


Fig. 1 Comparison of the risk-averse model to its deterministic and risk-neutral counterparts.

Table 3 The risk-averse model (tabu search) versus the risk-neutral model ($\alpha = 0.90$).

S	n = 10							
	DataSet 1		DataSet 2		DataSet 3		DataSet 4	
	θ	E(TWT)	θ	E(TWT)	θ	E(TWT)	θ	E(TWT)
100	-17.44%	3.08%	-14.82%	0.13%	-13.37%	5.54%	-21.39%	7.93%
150	-5.64%	21.21%	-9.72%	12.07%	-21.25%	9.10%	-8.73%	15.27%
200	-28.07%	5.40%	-7.27%	24.64%	-25.25%	13.84%	-15.83%	27.94%
250	-4.11%	0.32%	-5.40%	1.86%	-5.89%	0.39%	-11.32%	11.11%
300	-6.81%	18.77%	-5.47%	12.63%	-10.02%	6.38%	-11.90%	5.02%

S	n = 20							
	DataSet 1		DataSet 2		DataSet 3		DataSet 4	
	θ	E(TWT)	θ	E(TWT)	θ	E(TWT)	θ	E(TWT)
100	-17.98%	7.10%	-5.14%	16.99%	-25.77%	12.41%	-6.22%	15.27%
150	-15.81%	4.02%	-23.89%	0.88%	-12.07%	4.69%	-9.09%	5.36%
200	-2.62%	8.12%	-4.03%	4.06%	-2.99%	8.72%	-12.33%	22.70%
250	-3.87%	8.75%	-14.02%	12.81%	-5.16%	13.08%	-8.97%	5.93%
300	-14.19%	4.65%	-6.13%	7.33%	-12.97%	9.48%	-9.51%	11.43%

5 Conclusion

In this paper, we modeled the problem of minimizing VaR in the single-machine TWT problem under the presence of uncertainty and illustrated the value of the proposed risk-averse model. Furthermore, we presented a promising lower bounding scheme. As part of our ongoing research, we focus on developing an efficient mathematical programming based method to solve the proposed model VaR-TWT. In particular, we intend to incorporate a Lagrangian method to decompose the problem over scenarios.

References

- Alouloua MA, Croce FD (2008) Complexity of single machine scheduling problems under scenario-based uncertainty. *Operations Research Letters* 36:338–342
- Artzner P, Delbaen F, Eber JM, Heath D (1999) Coherent measures of risk. *Mathematical Finance* 9:203–227
- Bulbul K, Kaminsky P, Yano C (2007) Preemption in single machine earliness/tardiness scheduling. *Journal of Scheduling* 10(4-5):271–292
- Charnes A, Cooper W, Symonds G (1958) Cost horizons and certainty equivalents: An approach to stochastic programming of heating oil. *Management Science* 4:235–263
- Daniels R, Kouvelis P (1995) Robust scheduling to hedge against processing time uncertainty in single stage production. *Management Science* 41:363–376
- Dentcheva D (2006) Probabilistic and Randomized Methods for Design under Uncertainty, Springer-Verlag, London, chap Optimization Models with Probabilistic Constraints. Editor: Calafiore, G. and Dabbene, F.
- de Farias IR JR, Zhao H, Zhao M (2010) A family of inequalities valid for the robust single machine scheduling polyhedron. *Computers and Operations Research* 37:1610–1614
- Graham R, Lawler E, Lenstra J, Rinnooy Kan A (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5:287–326
- Gutjahr WJ, Hellmayr A, Pflug GC (1999) Optimal stochastic single-machine-tardiness scheduling by stochastic branch-and-bound. *European Journal of Operational Research* 117:396–413
- Kasperski A (2005) Minimizing maximal regret in the single machine sequencing problem with maximum lateness criterion. *Operations Research Letters* 33:431–436
- Kataoka S (1963) A stochastic programming model. *Econometrica* 31:181–196
- Keha AB, Khowala K, Fowler JW (2009) Mixed integer programming formulations for single machine scheduling problems. *Computers and Industrial Engineering* 56(1):357–367
- Larsen N, Mausser H, Uryasev S (2002) Financial Engineering, e-commerce and Supply Chain, Kluwer Academic Publishers, Berlin, chap Algorithms for Optimization of Value-at-Risk, pp 129–157. URL <http://www.ise.ufl.edu/uryasev/VaR-minimization.pdf>, P. Pardalos and V. K. Tsitsiringos (Eds.)
- Lehmann E (1955) Ordered families of distributions. *Annals of Mathematical Statistics* 26:399–419
- Lenstra J, Rinnooy Kan A, Brucker P (1977) Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1:343–362

- Mann HB, Whitney DR (1947) On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics* 18:50–60
- Muller A, Stoyan D (2002) *Comparison methods for stochastic models and risks*. Wiley, New York
- Nemhauser GL, Savelsbergh MWP (1992) *Combinatorial Optimization: New Frontiers in the Theory and Practice*, volume 82 of NATO ASI Series F: Computer and System Sciences, Springer, Berlin, chap A cutting plane algorithm for the single machine scheduling problem with release times, pp 63–84. M. Akgül, H. Hamacher, and S. Tufekci (Eds.)
- Ogryczak W, Ruszczyński A (1999) From stochastic dominance to mean-risk models: semideviations as risk measures. *European Journal of Operational Research* 116:33–50
- Ogryczak W, Ruszczyński A (2002) Dual stochastic dominance and related mean-risks models. *SIAM Journal of Optimization* 13(2):60–78
- Pan Y, Shi L (2007) On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems. *Mathematical Programming* 110:543–559
- de Panne CV, Popp W (1963) Minimum cost cattle feed under probabilistic constraints. *Management Science* 9:405–430
- Pinedo M (2008) *Scheduling: Theory, Algorithms, and Systems*, 3rd edn. Springer
- Potts C, van Wassenhove L (1982) A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters* 1(5):177–181
- Prékopa A (1995) *Stochastic Programming*. Kluwer Academic, Dordrecht, Boston
- Sen H (2010) A simple, fast, and effective heuristic for the single-machine total weighted tardiness problem. Master's thesis, Sabancı University, Istanbul, Turkey
- Sen T, Sulek JM, Dileepan P (2003) Static scheduling research to minimize weighted and unweighted tardiness: A state-of-the-art survey. *International Journal of Production Economics* 83(1):1–12
- Srinivasan V (1971) A hybrid algorithm for the one machine sequencing problem to minimize total tardiness. *Naval Research Logistics Quarterly* 18(3):317–327
- Tanaka S, Fujikuma S, Araki M (2009) An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling* 12:575–593
- Yang J, Yu G (2002) On the robust single machine scheduling problem. *Journal of Combinatorial Optimization* 6:17–33

Minimizing the maximal lateness on a single-machine: a new polynomially solvable case

Nodari Vakhania

Abstract Scheduling jobs with release times and due-dates on a single machine with the objective to minimize the maximal job lateness is known to be strongly NP-hard. The problem can be naturally simplified by imposing some restrictions on job processing times. Two such versions are known to be polynomially solvable: When all jobs have equal length and when a job processing time can be either p or $2p$, for some integer p . A straightforward generalization of the latter version with processing times $p, 2p, 3p, 4p$, etc., does not look too promising. On contrary, the version with mutually divisible processing times, e.g. $p, 2p, 4p, 8p$, etc., looks much more encouraging. We give a polynomial algorithm that solves the latter version. The algorithm is based on the general framework that reduces our problem to a version of the bin packing problem.

Key words: scheduling single machine, release time, due-date, mutually divisible processing times, lateness, bin packing

1 Introduction

Scheduling jobs with release times and due-dates on a single machine with the objective to minimize the maximal job lateness (with a common abbreviation $1/r_j/L_{\max}$) is known to be a strongly NP-hard problem [5]. Hence, it is unlikely that it can be solved in reasonable time in the general setting. In this problem n jobs with (non-interruptible) processing times p_j , release times r_j and due-dates d_j are to be scheduled on a single machine that can process at most one job at a time. Job parameters are non-negative integral numbers. The objective is to minimize the maximal job lateness, i.e., the difference between the actual completion time and the due-date of the job. The problem can be naturally simplified by imposing some restrictions on job processing times. Two such versions are known to be polynomially solvable: When all jobs have equal integer length p ($1/p_j = p, r_j/L_{\max}$) Garey et al [6] and when a job processing time can be either p or $2p$ ($1/p_j \in \{p, 2p\}, r_j/L_{\max}$) [12]. A straightforward generalization of the latter version with processing times $p, 2p, 3p, 4p$, etc., does not look too promising. On contrary, the version with *mutually divisible* processing times $1/p_j : \text{divisible}, r_j/L_{\max}$

Science faculty, State University Morelos, Mexico.
Partially supported by CONACyT grant 48433.
E-mail: nodari@uaem.mx

is more encouraging. In this version, job processing times can be ordered in a non-decreasing sequence such that for each two neighboring elements the second exactly divides the first. For example, the sequence can have the form $p, 2p, 4p, 8p$, etc. An example of a practical problem in which item sizes are mutually divisible arises in a computer memory allocation where block sizes are powers of 2. We give a polynomial time algorithm that solves $1/p_j : \text{divisible}, r_j/L_{\max}$. The algorithm is based on the general framework that reduces our problem to a version of the bin packing problem.

Baker and Su [1], Bratley et al [2], Carlier [3] and McMahon and Florian [8] have suggested exponential enumerative algorithms for $1/r_j/L_{\max}$. It is worth of mentioning that algorithms from [8] and [3] have a surprising good practical performance. Without release times, scheduling jobs in order of non-decreasing due-dates gives an optimal schedule in $O(n \log n)$ time Jackson [7]; similarly, if all d_{js} are equal, then scheduling jobs in order of non-decreasing release times is optimal. Other restrictions concern job processing times. The already mentioned algorithm from Garey et. al. [6] for equal-length job has time complexity of $O(n \log n)$. In this algorithm a concept of the so-called “forbidden region” is introduced. A forbidden region is a time interval in a schedule in which it is forbidden to start any job. In the algorithm the forbidden regions are defined, and a failure is declared if there exists no feasible schedule. Otherwise, a feasible schedule is generated, using the earliest deadline scheduling heuristic and the declared forbidden regions. Although this yields an $O(n^2 \log n)$ time complexity, the complexity is improved to $O(n \log n)$ by choosing a sophisticated data structures. The algorithm from [12] for $1/p_j \in \{p, 2p\}, r_j/L_{\max}$ has time complexity of $O(n^2 \log n \log p)$, and is obtained as a consequence of two enumerative algorithms for the version of the problem with arbitrary processing times. The first algorithm is exponential, whereas the second one, which is a restriction of the first one, runs in $O(n^2 \log n)$ time if certain conditions during its execution are satisfied. Otherwise, a so-called Balancing Procedure is to be applied to guarantee the optimality. As it is shown, there is an $O(n^2 \log n \log p)$ implementation of Balancing Procedure for $1/p_j \in \{p, 2p\}, r_j/L_{\max}$, which yields an overall algorithm with the same time complexity.

The multiprocessor case with m parallel identical processors can also be solved polynomially if job processing times are equal. Simons & Warmuth [10] have suggested an $O(n^2 m)$ algorithm which uses the earlier mentioned forbidden regions from [6] to find a schedule that meets all job due-dates. It applies the earliest deadline scheduling rule for the construction of such feasible schedules, while each time the next scheduled job fails to meet its deadline, backtracking is performed and a new forbidden region is declared. It is easily seen that the minimization problem can be solved by the repeated application of this algorithm for the corresponding feasibility problem: we can iteratively increase the due-dates of all jobs until we find a feasible schedule with the modified data. Since the maximum tardiness will depend on the job processing time and the number of jobs, we may need to apply such an algorithm np/m times (p is the job processing time). This number can be reduced to $\log(np/m)$ if we use a binary search. In [11] was later proposed an $O(d_{\max} mn \log n + O(m\kappa n))$ algorithm for the minimization version, where $\kappa < n$ is a parameter which is known after the termination of the algorithm (in practice, κ turns out to be a small positive integer). The algorithm enumerates a special type of feasible schedules called the *complementary schedules* on a search tree. A complementary schedule is constructed by the earliest due-date heuristic (ED-H), proposed earlier in [7] and [9]. This heuristic is repeatedly applied to the iteratively modified problem instances and different complementary schedules are produced. In each generated complementary schedule a new gap (a time interval in a

schedule, not occupied by a job) arises. These gaps are somewhat similar to the earlier mentioned forbidden regions (though no special computational effort for their construction is needed). The so called overflow job realizes the maximal value of our objective function in a schedule. The behavior alternatives reflect all possible ways of altering of the overflow job in a newly generated complementary schedule. The algorithm is based on a special analysis of behavior alternatives.

The framework that we present in this paper yields an $O(n^2 \log n \log p_{\max})$ algorithm for $1/p_j : \text{divisible}, r_j/L_{\max}$. The algorithm uses some mentioned concepts introduced in [11] and [12] incorporating though a different type of a search. In particular, it does not conduct the search on a solution tree but it uses binary search. The whole set of jobs is partitioned into two kinds of subsets, non-critical and critical. The non-critical subsets contain jobs that might be flexibly moved within the schedule, whereas the critical sets that we call kernels contain the subsets of jobs which form tight sequences in the sense that the delay of the earliest scheduled job from the subset cannot exceed some calculated parameter between (including) 0 and $p_{\max}$. When the delay of the latter job is 0, the lateness of the latest scheduled job from the set defines a valid lower bound on the optimal value. We define the set of kernels and then determine the above lower bounds yielded by each kernel. The maximum among them is a valid lower bound for the problem, and it also delineates the maximal delay that might be imposed to other kernels without increasing the maximum lateness. Having at hand all the above magnitudes, we try to distribute non-kernel jobs in order to utilize the intervals in between kernels that we call bins, in an optimal way.

We came to a related version of bin packing problem: we have a fixed number of bins of different capacities and we wish to know if the given items (non-kernel jobs) can be distributed into these bins. Our task is more complicated though. Due to job release times and due dates, not any non-kernel job can be placed into any bin, besides bin sizes have some degree of flexibility. Even without these restrictions this version of bin packing problem is known to be NP-hard. Hence we don't really expect to arrive to a polynomial-time solution for the general $1/r_j/L_{\max}$. Here it is worth of mentioning a venerable paper by Coffman, Garey & Johnson [4] that studies different versions of bin packing problem when the item sizes are mutually divisible. The authors show that these versions can be solved in polynomial time. The same restriction on job processing times results in a polynomial time solution in our scheduling problem. It is sufficient to restrict the processing times of only non-critical jobs; the processing times of the jobs from the critical subsets can, in fact, be arbitrary: in some applications if it is predictable which jobs will form a critical sequence, then the processing times of such jobs can be set without any restriction). Applying a well-known ED-heuristic to the original problem instance we obtain an upper bound on the delay for the critical sequences. Since the minimal possible delay is 0, we may carry out binary search to find the delay that would result in an optimal schedule.

2 Defining rigid segments

Let us first define a *feasible schedule* S as a mapping that assigns to each job j a starting time $t_j(S)$, such that $t_j(S) \geq r_j$ and $t_j(S) \geq t_k(S) + p_k$, for any job k included earlier in S ; the first inequality says that a job cannot be started before its release time, and the second one reflects the restriction that the machine can handle only one job at any time. We denote by $c_j(S) = t_j(S) + p_j$ the completion time of job j . We aim to find

out if there is a schedule which meets all job due-dates, i.e., every j is completed by time d_j . If there is no such schedule then we aim to find an optimal schedule, i.e., one minimizing the maximal *lateness* $L_{\max} = \max\{j|c_j - d_j\}$.

Denote by $f(S)$ ($f_S(j)$, respectively) the maximal lateness in S (the lateness of $j \in S$, respectively). To classify the jobs, we construct our first feasible schedule using *Earliest Due-date* heuristic (ED-H) suggested by Jackson [7] in early 1955. The preemptive (on-line) version of this heuristic optimally solves the preemptive version of our (general) problem: Starting from time 0, at any time t , among all available jobs, schedule next one with the smallest due-date; if a job with a due-date, smaller than the currently processed job becomes available, interrupt the latter job and schedule the former job (note that the resulting schedule will have no machine idle time unless no job is available). For non-preemptive case, assume at time moment r_j job i is processed where $d_j < d_i$. As i cannot be preempted it will delay the starting of job j . Note that this delay will be less than p_i . The delay of job j in S (equal to the magnitude $t_j^S - r_j$) may result a non-optimal schedule (it may turn out that j has a tight due-date and had to be scheduled without or with less delay in an optimal schedule).

Let t be the maximum between the minimal release time of yet unscheduled job and the time when the machine completes the latest scheduled job (0 if no job is yet scheduled). Initially, we have no scheduled jobs. Iteratively, ED-H (the non-preemptive version of the above mentioned preemptive heuristic), among all available jobs at time t , schedules a job with the smallest due-date breaking ties by selecting a longest job. The *initial ED-schedule* σ is the one generated by ED-H for the originally given problem instance. As we have noted, the value of our objective function for the latter ED-schedule may be $p_{\max} = \max\{p_i | i = 1, 2, \dots, n\} - 1$ more than the optimal value (this absolute error delineates the frontier between the polynomial solvability and strong NP-completeness).

A *gap* in a schedule is a maximal consecutive time interval in which no job is processed by the machine. By our convention, there occurs a 0-length gap (c_j, t_i) whenever job i starts at its release time immediately after the completion of job j . A *block* in an ED-schedule S is its consecutive part preceded and succeeded by a (possibly a 0-length) gap.

Now we define critical and non-critical jobs introduced informally in the introduction. Our primary goal now is to determine rigid (critical) segments in our initial ED-schedule σ . This we do by simply verifying $f_\sigma(j)$ for each included j . In general, we call a job o in S an *overflow job* if $f_S(o) = \max\{f(j) | j \in S\}$ (we note that $f_S(o)$ might be positive or non-positive magnitude: in the former case there arises a late job in S).

We call a *kernel* a maximal job sequence/set in S ending with an overflow job o such that no job from this sequence has a due-date more than d_o ; if there are several successively scheduled overflow jobs then o is the latest one. Note that kernel jobs imply rigid segments in a feasible schedule, i.e., they are to be scheduled within a highly restricted time interval. Let $\mathcal{K}$ be the set of all kernels $K_1 \prec K_2 \prec \dots \prec K_k$ arisen in σ , and let $r(K) = \min_{i \in K} \{r_i\}$, for a kernel K . If the earliest scheduled kernel job starts at time $r(K)$ then there is no feasible schedule S' with $f(S') < f_S(o)$ (as reordering kernel jobs cannot reduce the lateness) and S is an optimal schedule:

Observation 1 *An ED-schedule S is optimal if it contains a kernel with its earliest scheduled job starting at its release time.*

Assume for this rest of the paper that the condition in Observation 1 does not hold. Then there is job scheduled before all kernel jobs that delays kernel jobs including the overflow job. By removing such a job the kernel jobs might be restarted earlier reducing in this way $f_{\max}$. To be more formal, let us introduce some definitions. Suppose i precedes j in an ED-schedule S . We will say that i *pushes* j in S if j gets rescheduled earlier whenever i is removed and the succeeding jobs from S are rescheduled as early as possible respecting the order in S .

It follows from our assumption and definitions that the earliest scheduled job of every kernel is immediately preceded and pushed by a job e with $d_e > d_o$. Clearly, we may have more than one such a job scheduled before kernel K in the block containing K . We call such a job an *emerging job* for K , and the latest scheduled one (job e above) the *delaying emerging job*. We shall later refer to j with $d_j > d_o$ and $r_j < r(K)$ scheduled after K as a *passive emerging job* for K .

To restart the kernel jobs earlier, we *activate* an emerging job e for K , *that is*, we force it and all passive emerging jobs to be rescheduled after K by increasing their release times to a sufficiently large magnitude, say $r(K)$ (the latter jobs also are said to be activated for K). Then, when ED-H is again applied, neither job e nor any passive emerging job will surpass any kernel job and hence the earliest job in K *will start* at $r(K)$. In general, more than one emerging job might be activated for K and the same emerging job may be activated for two or more successive kernels.

3 Basis for the binary search procedure

δ -balanced schedules. Consider an (incomplete) ED-schedule σ^{**} in which the delay job of every $K \in \mathcal{K}$ is just omitted, and let $f'(i)$ be the new (reduced) value of the lateness of each kernel job i in σ^{**} . Since every K is (re)started at time $r(K)$ in σ^{**} , $L(K) = \max_{i \in K} \{f'(i)\}$ is a lower bound on the value of the optimal schedule. For any feasible S , $f(S) \geq L^* = \max_{\kappa} \{L(K_{\kappa})\}$ is a stronger lower bound. Furthermore, if $\delta(K) = L^* - L(K)$, then clearly, in any feasible S we may allow the delay of $\delta(K) \geq 0$ without increasing the current maximal lateness, for every K .

We still need to accommodate every delaying job omitted in σ^{**} . Assume the delaying job is activated for each K . This simple solution might be too rough as due to the gaps left before each kernel, no available space might be left for scheduling the activated jobs together with the rest of the jobs. So the gap(s) before each kernel are to be somehow beneficially used. We use our binary search procedure. We know that the earliest scheduled job of every K can be started at time $r(K) + \delta(K)$. In an optimal schedule S_{opt} , either each kernel K starts no later than at time $r(K) + \delta(K)$ or K is to be delayed by some δ , $0 \leq \delta \leq \Delta$, where $\Delta = f(o) - L^*$. Observe that if the earliest job of every K starts no later than at time $r(K) + \delta(K) + \delta$ then $f(i) \leq L^* + \delta$, for any $i \in K$. More generally, we call a feasible schedule $S(\delta)$ with $f(S(\delta)) \leq L^* + \delta$ *δ -balanced* (note that $\sigma = S(\Delta)$). The magnitude $L^* + \delta$ is our *δ -boundary*; job j *surpasses* the δ -boundary if $f(j) > L^* + \delta$.

Thus we allow an extra delay of δ for any (former) overflow job in $S(\delta)$, whereas the first job of every kernel $K \in S(\delta)$ starts no later than at time $r(K) + \delta(K) + \delta$ (we respectively redefine the delaying job for a kernel K as one that completes *after* time $r(K) + \delta(K) + \delta$). Though, this doesn't yet guarantee that no job from K will surpass the δ -boundary when ED-H is again applied. This will be clearly so if K starts at moment $r(K) + \delta(K) + \delta$.

Suppose the earliest job of K starts strictly earlier at moment $r(K) + \delta(K) + \delta - \epsilon$, for an integer $\epsilon \geq 1$. Then when scheduling jobs of K by ED-H, there may occur a time moment at which some kernel job completes but no yet unscheduled job from K is released. Then some external job might be included at that (or later) moment. We put a restriction on the total length of such external jobs that might be included in between the jobs of K (more formally, in any feasible S with $f(S) \leq L(K) + \delta(K) + \delta$ the above length is to be restricted as follows): clearly, if no gap in between jobs of K occurs then the above magnitude cannot be more than ϵ , otherwise it cannot exceed ϵ minus the total length of the gap(s) occurred. We shall refer to this restriction as the *fitness* rule for K . Based on the next lemma which immediately follows, we further assume that this rule is respected while scheduling each kernel in $S(\delta)$:

Lemma 1 *No job from any kernel K will surpass the δ -boundary if the fitness rule is respected.*

Although no job from the current set of kernels may surpass the current δ -boundary, there may arise another job that surpasses it. We wish to find out if there exists $S(\delta)$. On the first iteration of our binary search procedure when we generate $\sigma = S(\Delta)$, $\delta = \Delta$. The next value for δ is 0; if there exists no $S(0)$ then the next value of δ is $\lceil \Delta/2 \rceil$. So δ is derived from the interval $[0, \Delta]$, whereas the change from larger to smaller value of δ is carried out if a δ -balanced schedule for the current δ was successfully created; otherwise, δ is increased respectively on the next iteration. The following observation is apparent:

Observation 2 *$S(\delta)$ with minimal possible δ is optimal.*

Observe now that our problem is already solved given that we have a procedure that either constructs a $S(\delta)$ or asserts that it doesn't exist. As $\delta < p_{\max}$, the number of iterations for the binary search procedure is bounded by $\log p_{\max}$. Then roughly, the running time of the overall algorithm would be $\log p_{\max}$ multiplied by the time complexity of the above procedure.

4 Seeking after $S(\delta)$

The procedure $\text{SEEK}(S(\delta))$ that seeks after $S(\delta)$ consists of several passes. It uses ED-rule in which ties are broken by selecting a longest job. Denote by $\mathcal{K}_\delta$ the set of kernels corresponding to δ in the binary search procedure (the initial set of kernels $\mathcal{K} = \mathcal{K}_\Delta$). For $\delta' < \delta''$, $|\mathcal{K}_{\delta'}| \geq |\mathcal{K}_{\delta''}|$. Whenever the binary search procedure resumes with a new δ , $\text{SEEK}(S(\delta))$ sets initially $\mathcal{K}_\delta$ to $\mathcal{K}'_\delta$, where $\delta' > \delta$ is the smallest so far encountered value with an existing $S(\delta')$. As $\text{SEEK}(S(\delta))$ advances in forming a complete schedule, the current $\mathcal{K}_\delta$ might be completed by a new kernel. Recall that each $K \in \mathcal{K}_\delta$ starts no later than at time $r(K) + \delta(K) + \delta$ in $S(\delta)$. $\text{SEEK}(S(\delta))$ respects this restriction. Hence if there arises a job j surpassing the δ -boundary then it cannot belong to any kernel in $\mathcal{K}_\delta$ (see also Lemma 1).

We shall refer to the interval before each $K \in \mathcal{K}_\delta$ as the *bin* defined by K and denote it by B_K . We may have a non-idle time interval after the latest kernel of $\mathcal{K}_\delta$, that we also call bin and denote by B_+ .

Observe that if $\text{SEEK}(S(\delta))$ has succeeded to construct a complete schedule so that no bin job has surpassed the δ -boundary, then this schedule is $S(\delta)$. Assume K was the latest scheduled kernel from $\mathcal{K}_\delta$ when there has occurred (a non-kernel job) j

surpassing the δ -boundary. If j is a former emerging job (one activated for K or/and some preceding kernel) then we will say that an *instance of alternative* (b1) (IA(b1)) with job j occurs.

If j is not a former emerging job, then an activated (former emerging) job must be pushing j . If among such jobs there is an emerging job for j , let e be the latest scheduled one. If e was included before K , then the jobs from K together with j and all jobs that were included after e (before j has occurred) define a new kernel, also denoted by K . If e was included after K , then the sequence of jobs in between e and j (including j) forms a new kernel K' for the current δ . SEEK($S(\delta)$) updates the current $\mathcal{K}_\delta$ respectively.

If no new kernel can be defined, i.e., there is no e , let i be an activated (former emerging) job pushing j . Then an *instance of alternative* (b2) (IA(b2)) with job i is said to occur.

It immediately follows that if there has arisen a non-kernel job surpassing the δ -boundary then there must be occurring an IA(b1/b2). Hence, if no IA(b1/b2) occurs then we already have a correct answer (for the general $1/r_j/L_{\max}$). Otherwise, to complete the description of SEEK($S(\delta)$) we need to describe how we rearrange non-kernel jobs for an IA(b1/b2). Clearly, at least one passive emerging job q for K is to be rescheduled before K . This will not be possible (in $S(\delta)$), unless some job i scheduled in B_K or some earlier bin is rescheduled after K (if this were possible, ED-H would include q in B_K). More generally, suppose we impose the delay of jobs $i_1, i_2, \dots, i_\ell$ so that they are not scheduled before K . In the resultant schedule portion before K , let γ be a gap occurring after time r_q , and λ be the total length of all gaps including γ and all succeeding gaps before K . Then γ is (*potentially*) *valid* for q (subject to $i_1, i_2, \dots, i_\ell$) if $p_q < \lambda$. We call job s pushing q a *substitution* job for q if it is an emerging job for K (s is from B_K or some earlier bin). Suppose we activate some substitution job(s) for K . If among the newly arisen gaps there is a valid one for q subject to the above substitution job(s) then, and only then, it might be possible to avoid another IA(b1/b2):

Observation 3 *Suppose there occurs an IA(b1/b2) behind kernel K . Then there exists no $S(\delta)$ if there arises no valid gap for none of the passive emerging jobs for K subject to some substitution jobs.*

Proof. First we show that the claim holds if there exists no substitution job for none of the passive emerging jobs. At least one passive emerging job q for K is to be rescheduled before K ; hence some job i scheduled before K is to be moved after K . But since i is not a substitution job for q , it is not an emerging job for K . Then it will surpass the δ -boundary. Now, if there arises no valid gap for q subject to some substitution jobs then similarly, the δ -boundary will be surpassed. $\square$

Thus all SEEK($S(\delta)$) has to do is to activate substitution jobs for K in a proper fashion, whenever IA(b1/b2) occurs. Such a valid polynomial-time procedure when job processing times are arbitrary is unlikely to exist. We describe how SEEK($S(\delta)$) guarantees a correct answer when the processing times of non-kernel jobs are mutually divisible. As powers of 2 is the most “dance” such a set, let us assume that the processing time of a non-kernel job is a power of 2.

Suppose we have an IA(b1/b2) with job j , j and K being defined as before. We know that some substitution job(s) are to be activated for K . By ED-H, all potentially useful substitution jobs must have been scheduled in the same bin $B(K')$, each of them being started strictly before the moment r_j , where $K' = K$ or $K \prec K'$. Denote this

set of these substitution jobs by $\text{SUBST}(j, \delta)$, and denote the corresponding set of the passive emerging jobs by $\text{PASS}(j)$ (note that $j \in \text{PASS}(j)$). Basically, we need to determine which jobs from $\text{SUBST}(j, \delta)$ are to be activated for K , whereas ED-H will determine the job(s) in $\text{PASS}(j)$ which will be included within the released space in $B(K')$. The total length of these jobs must be “enough long” to provide a sufficient left-shift to avoid another IA(b1/b2) with a job from $\text{PASS}(j)$. We do not bother ourselves to determine this length exactly (though there is an easily seen upper bound of $p_{\max}$ on this length).

By ED-H, job j cannot be released before the latest scheduled job in $\text{SUBST}(j, \delta)$ is started. Hence, the jobs in $\text{SUBST}(j, \delta)$ are scheduled one after another as a single sequence. Denote the interval occupied by this sequence by $I(\text{SUBST}(j, \delta))$. We may achieve a desired left-shift for the job(s) in $\text{PASS}(j)$ by rescheduling some job(s) from $\text{SUBST}(j, \delta)$ behind K . Intuitively, a subset of $\text{SUBST}(j, \delta)$ with a correct total length would work out: this length should be enough to left-shift job j by a sufficient amount of time units; though, if it is “too long it may cause a space shortage at a later schedule portion.

We activate jobs from $\text{SUBST}(j, \delta)$ for K one by one, the latest scheduled ones first, starting with the latest scheduled job from $\text{SUBST}(j, \delta)$. We activate the next job from $\text{SUBST}(j, \delta)$ whenever the already accomplished left-shift is not yet enough; that is, there again occurs IA(b1/b2) with a job from $\text{PASS}(j)$. Obviously, if there are jobs in $\text{SUBST}(j, \delta)$ with a “sufficient” total length then once the next substitution job is activated no more IA(b1/b2) with a job from $\text{PASS}(j)$ will occur. Otherwise, if no more substitution job remains then, clearly, there exists no δ -balanced schedule and $\text{SEEK}(S(\delta))$ stops with a “no” answer. Suppose $\text{SEEK}(S(\delta))$ arrives to a stage when after the activation of the next substitution job no further IA(b1/b2) with a job from $\text{PASS}(j)$ occurs. If a complete schedule is generated so that no IA(b1/b2) with a job from $\text{SUBST}(j, \delta)$ arises then the arrangement of jobs in $\text{SUBST}(j, \delta)$ and $\text{PASS}(j)$ is obviously optimal. Then $\text{SEEK}(S(\delta))$ repeats the above procedure defining the sets $\text{SUBST}(j', \delta)$ and $\text{PASS}(j')$, for the next arisen job j' surpassing the δ -boundary. If no j' occurs then $\text{SEEK}(S(\delta))$ halts with a δ -balanced schedule.

So, it still remains to consider the case when IA(b1/b2) with a job $s \in \text{SUBST}(j, \delta)$ occurs. Before doing into the details, observe that the rest (already described part) of $\text{SEEK}(S(\delta))$ is valid for the general $1/r_j/L_{\max}$. To guarantee the optimality, we shall now take an advantage of mutually divisible job processing times. We know that job s is to be left-shifted to avoid IA(b1/b2) with s . For this, either s itself or some preceding to s substitution job from $\text{SUBST}(j, \delta)$ is to be rescheduled (back) within the interval $I(\text{SUBST}(j, \delta))$, and clearly, this is the only possibility. Recall that $\text{SEEK}(S(\delta))$ has rescheduled some passive emerging job(s) within $I(\text{SUBST}(j, \delta))$ on the previously described step. Now, among the already activated substitution jobs from $\text{SUBST}(j, \delta)$, we determine the jobs that can fit into the remained available space within $I(\text{SUBST}(j, \delta))$ and reschedule them back (breaking ties by selecting more urgent jobs). Once this operation is done, there remains no available (short enough) job that can be inserted within $I(\text{SUBST}(j, \delta))$. Let $c \geq p_{\min}$ be the maximal length of a job that can be inserted in $B(K')$ (without forcing other jobs to surpass the δ -boundary), and let p be the maximal job processing time such that $p \leq c$. We know that there is no job of length p or less that can be inserted in $B(K')$. However, the total length of jobs that may potentially be scheduled in $B(K')$ may still be more than the total length of jobs currently scheduled in $B(K')$. This may potentially be achieved by interchanging some jobs from $\text{SUBST}(j, \delta)$ (still from $I(\text{SUBST}(j, \delta))$ by some other jobs from $\text{SUBST}(j, \delta)$

already activated for K . But this is not possible if jobs are mutually divisible. This follows from the next lemma which, in turn, can be straightforwardly seen:

Lemma 2 *Suppose we have a collection of jobs with processing times $2p, 4p, 8p, \dots, kp$, where k is a power of 2. Then no sequence containing some subset of these jobs scheduled without a gap can be completed at time $(k-1)p$, more generally, at any time moment ιp , for an odd ι .*

It follows that if among the jobs from $\text{SUBST}(j, \delta)$ remained in $I(\text{SUBST}(j, \delta))$ there is no job less urgent than s then no δ -balanced schedule exists. Otherwise, we replace s with the latest scheduled such a job. And we repeat the same procedure if a repeated $\text{IA}(b1/b2)$ occurs.

This completes the description of $\text{SEEK}(S(\delta))$ and the proof of its soundness. It is easily seen that the time complexity of $\text{SEEK}(S(\delta))$ is $n^2 \log n$. Indeed, the ED-H with the cost $n \log n$ is used less than n times while activating each substitution job for an $\text{IA}(b1/b2)$. Hence, the time complexity of our binary search procedure is $n^2 \log n \log p_{\max}$:

Theorem 1 *The binary search procedure finds an optimal schedule in time $O(n^2 \log n \log p_{\max})$.*

5 Final remarks

We have proposed a general framework that reduces $1/r_j/L_{\max}$ to a version of bin packing problem using a binary search procedure. We have shown that the cost of this procedure is polynomial for $1/p_j : \text{divisible}, r_j/L_{\max}$. The main question that remains is whether $1/p_j : \text{divisible}, r_j/L_{\max}$ is a maximal polynomially solvable special case of $1/r_j/L_{\max}$ with restricted job processing times. If it is not, is the proposed framework good enough to be used for a version with a less restricted job processing times?

References

1. K.R. Baker and Zaw-Sing Su (1974). Sequencing with due-dates and early start times to minimize maximum tardiness. *Naval Res. logist. Quart.* 21, 171–177.
2. P. Bratley, M. Florian and P. Robillard (1973). On sequencing with earliest start times and due-dates with application to computing bounds for $(n/m/G/F_{\max})$ problem. *Naval Res. logist. Quart.* 20, 57–67.
3. J. Carlier (1982). The one-machine sequencing problem. *European J. of Operational Research.* 11, 42–47.
4. Coffman E.G. Jr., M.R. Garey and D.S. Johnson. Bin packing with divisible item sizes. *Journal of Complexity* 3, 406–428 (1987)
5. Garey M.R. and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-completeness (Freeman, San Francisco, 1979)
6. Garey M.R., D.S. Johnson, B.B. Simons and R.E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM J. Comput.* 10, 256–269 (1981)
7. Jackson J.R. Scheduling a production line to minimize the maximum lateness. *Management Science Research Report 43, University of California, Los Angeles* (1955)
8. G. McMahon and M. Florian. On scheduling with ready times and due-dates to minimize maximum lateness. *Operations Research.* 23, 475–482 (1975)
9. Schrage L. Obtaining optimal solutions to resource constrained network scheduling problems, unpublished manuscript (march, 1971)
10. Simons B., M. Warmuth. A fast algorithm for multiprocessor scheduling of unit-length jobs. *SIAM J. Comput.* 18, 690–710 (1989)

11. Vakhania N. A better algorithm for sequencing with release and delivery times on identical processors. *Journal of Algorithms* 48, p.273-293 (2003)
12. Vakhania N. "Single-Machine Scheduling with Release Times and Tails". *Annals of Operations Research*, 129, p.253-271 (2004)

Minimizing the sum of quadratic completion times on two identical parallel machines

Federico Della Croce · Christos Koulamas

Abstract We discuss the approximation ratio of the *SPT* rule for the problem of minimizing the sum of quadratic completion times on two parallel machines. We show that the approximation ratio is bounded above by $\frac{\sqrt{5}+2}{\sqrt{5}+1} \approx 1.309$ and below by $\frac{11+4\sqrt{5}}{11+2\sqrt{5}} \approx 1.289$.

Keywords: Parallel machines, list scheduling, quadratic completion times.

1 Introduction

We consider the problem of minimizing the sum of quadratic completion times on two parallel machines. The problem can be stated as follows. A set of n jobs must be processed on a set of two identical parallel machines. Each job i has a processing time p_i which is identical on all machines. Each machine can process one job at a time. Preemption is not allowed. The objective is to minimize the sum of quadratic completion times. The problem is usually referred to as $P2||\sum C_j^2$. The more general $P||\sum C_j^2$ problem is known to be NP-hard in the strong sense ([1]). In [1], it is also shown by probabilistic analysis that the approximation ratio of the *SPT* rule (that always schedules the shortest unscheduled job whenever a machine becomes idle) asymptotically solves the problem. Notice that the *SPT* rule is optimal on a single machine [2]. Here, we show, that on 2 machines, the approximation ratio of the *SPT* rule is bounded above by $\frac{\sqrt{5}+2}{\sqrt{5}+1} \approx 1.309$. Also, we show that the approximation ratio is bounded below by $\frac{11+4\sqrt{5}}{11+2\sqrt{5}} \approx 1.289$.

F. Della Croce
D.A.I., Politecnico di Torino, Torino, Italy
E-mail: federico.dellacroce@polito.it

C. Koulamas
D.S.I.S., Florida International University, Miami, U.S.A.
E-mail: koulamas@fiu.edu

2 Main result

W.l.o.g. we assume jobs indexed in shortest processing time order, that is $p_1 \leq p_2 \leq \dots \leq p_n$ with ties broken arbitrarily. Given any schedule S , we denote by $C_j(S)$ the completion time of job j and by $C_{[j]}(S)$ the j -th completion time in the schedule S . Let S_{OPT} and S_{SPT} be the optimal schedule and the schedule obtained by applying the SPT rule respectively. The following lemma holds.

Lemma 1 *For any sequence S , we have $C_{[j]}(S) \geq \max\{p_j, \frac{\sum_{i=1}^j p_i}{2}\}$.*

Proof: $C_{[j]}(S) \geq p_j$ trivially holds as j is the j -th jobs in the SPT order. Besides,

$C_{[j]}(S) \geq \frac{\sum_{i=1}^j p_i}{2}$ holds as $\sum_{i=1}^j p_i$ is a lower bound on the sum of the processing times of the first j jobs in S and the best we can do is to equally partition these processing times on the two machines. $\square$

The following lemma also holds.

Lemma 2 *For any sequence S , we have $C_{[j]}(S) + C_{[j-1]}(S) \geq \sum_{i=1}^j p_i$.*

Proof: If $[j]$ and $[j-1]$ are on different machines,

$$C_{[j]}(S) + C_{[j-1]}(S) = \sum_{i=1}^j p_{[i]}(S) \geq \sum_{i=1}^j p_i.$$

Alternatively, $[j]$ and $[j-1]$ are on the same machine. Let $[h]$ be the largest completion time job on the other machine such that $C_{[h]}(S) \leq C_{[j-1]}(S)$. Then,

$$C_{[j]}(S) + C_{[h]}(S) = \sum_{i=1}^j p_{[i]}(S) \geq \sum_{i=1}^j p_i \text{ and, correspondingly,}$$

$$C_{[j]}(S) + C_{[j-1]}(S) \geq C_{[j]}(S) + C_{[h]}(S) \geq \sum_{i=1}^j p_i. \quad \square$$

We focus now on the completion times of the $(j-1)$ -th job and the j -th job in the SPT schedule. The following lemma holds.

Lemma 3 *In the SPT schedule, the following conditions hold:*

$$(a) C_{[j]}(S_{SPT}) + C_{[j-1]}(S_{SPT}) = \sum_{i=1}^j p_i,$$

$$(b) C_{[j]}(S_{SPT}) \leq p_j + C_{[j-1]}(S_{SPT}),$$

$$(c) C_{[j]}(S_{SPT}) \leq p_j + \frac{\sum_{i=1}^{j-1} p_i}{2}.$$

Proof: Given the SPT rule, the first condition has been shown in the proof of Lemma

2. For the second condition, given that

$$C_{[j]}(S_{SPT}) + C_{[j-1]}(S_{SPT}) = \sum_{i=1}^j p_i \text{ and correspondingly}$$

$$C_{[j-1]}(S_{SPT}) + C_{[j-2]}(S_{SPT}) = \sum_{i=1}^{j-1} p_i, \text{ we derive}$$

$$C_{[j]}(S_{SPT}) = C_{[j-2]}(S_{SPT}) + p_j \text{ with } C_{[j-2]}(S_{SPT}) \leq C_{[j-1]}(S_{SPT}). \text{ Hence,}$$

$$C_{[j]}(S_{SPT}) \leq C_{[j-1]}(S_{SPT}) + p_j.$$

For the third condition, as

$$C_{[j-1]}(S_{SPT}) + C_{[j-2]}(S_{SPT}) = \sum_{i=1}^{j-1} p_i \text{ and}$$

$$C_{[j-2]}(S_{SPT}) \leq C_{[j-1]}(S_{SPT}), \text{ we have}$$

$$C_{[j-2]}(S_{SPT}) \leq \frac{\sum_{i=1}^{j-1} p_i}{2}. \quad \square$$

The following Properties hold.

Property 1 *If $\sum_{i=1}^{j-1} p_i \geq p_j$, then $C_{[j]}^2(S_{OPT}) + C_{[j-1]}^2(S_{OPT}) \geq 2(\frac{\sum_{i=1}^j p_i}{2})^2$
else, $\sum_{i=1}^{j-1} p_i < p_j$ and $C_{[j]}^2(S_{OPT}) + C_{[j-1]}^2(S_{OPT}) \geq p_j^2 + (\sum_{i=1}^{j-1} p_i)^2$.*

Proof: Assume the first condition ($\sum_{i=1}^{j-1} p_i \geq p_j$) holds. From Lemma 1, we have $C_{[j]}(S_{OPT}) \geq \frac{\sum_{i=1}^j p_i}{2}$. Let $\alpha \geq 0$ such that $\alpha = C_{[j]}(S_{OPT}) - \frac{\sum_{i=1}^j p_i}{2}$. Correspondingly, $C_{[j]}(S_{OPT}) = \frac{\sum_{i=1}^j p_i}{2} + \alpha$.

From Lemma 2, we get $C_{[j-1]}(S_{OPT}) \geq \frac{\sum_{i=1}^j p_i}{2} - \alpha$. Summing up,

$$C_{[j]}^2(S_{OPT}) + C_{[j-1]}^2(S_{OPT}) \geq (\frac{\sum_{i=1}^j p_i}{2} + \alpha)^2 + (\frac{\sum_{i=1}^j p_i}{2} - \alpha)^2 =$$

$$= 2(\frac{\sum_{i=1}^j p_i}{2})^2 + 2\alpha^2 \geq 2(\frac{\sum_{i=1}^j p_i}{2})^2.$$

Now, assume the second condition ($\sum_{i=1}^{j-1} p_i < p_j$) holds. From Lemma 1, we have $C_{[j]}(S_{OPT}) \geq p_j$ and from Lemma 2, we have

$C_{[j]}(S_{OPT}) + C_{[j-1]}(S_{OPT}) \geq p_j + \sum_{i=1}^{j-1} p_i$.
 Let $\beta \geq 0$ such that $\beta = C_{[j]}(S_{OPT}) - p_j$. Correspondingly,
 $C_{[j]}(S_{OPT}) = p_j + \beta$ and $C_{[j-1]}(S_{OPT}) \geq \sum_{i=1}^{j-1} p_i - \beta$. Summing up,

$$C_{[j]}^2(S_{OPT}) + C_{[j-1]}^2(S_{OPT}) \geq (p_j + \beta)^2 + (\sum_{i=1}^{j-1} p_i - \beta)^2 =$$

$$= p_j^2 + (\sum_{i=1}^{j-1} p_i)^2 + 2\beta^2 + 2\beta p_j - 2\beta(\sum_{i=1}^{j-1} p_i) =$$

$$= p_j^2 + (\sum_{i=1}^{j-1} p_i)^2 + 2\beta(\beta + p_j - \sum_{i=1}^{j-1} p_i) \geq p_j^2 + (\sum_{i=1}^{j-1} p_i)^2$$
 as $p_j > \sum_{i=1}^{j-1} p_i$ and $\beta \geq 0$. $\square$

Property 2 If $\sum_{i=1}^{j-1} p_i \geq p_j \forall j$, then $C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT}) \leq \frac{5}{8}(\sum_{i=1}^j p_i)^2$.

Proof: From Lemma 3.a, we have

$C_{[j]}(S_{SPT}) + C_{[j-1]}(S_{SPT}) = \sum_{i=1}^j p_i$ with $C_{[j]}(S_{SPT}) \geq C_{[j-1]}(S_{SPT})$. Hence,
 $C_{[j]}(S_{SPT}) \geq \frac{\sum_{i=1}^j p_i}{2}$ and $C_{[j-1]}(S_{SPT}) \leq \frac{\sum_{i=1}^j p_i}{2}$, that is
 $C_{[j]}(S_{SPT}) = \frac{\sum_{i=1}^j p_i}{2} + \gamma$ and $C_{[j-1]}(S_{SPT}) = \frac{\sum_{i=1}^j p_i}{2} - \gamma$ with $\gamma \geq 0$. Hence,
 $C_{[j]}(S_{SPT}) - C_{[j-1]}(S_{SPT}) = 2\gamma$. Also, from Lemma 3.b, we have $2\gamma = C_{[j]}(S_{SPT}) - C_{[j-1]}(S_{SPT}) \leq p_j$, namely $\gamma \leq \frac{p_j}{2}$. Then,

$$C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT}) = (\frac{\sum_{i=1}^j p_i}{2} + \gamma)^2 + (\frac{\sum_{i=1}^j p_i}{2} - \gamma)^2 =$$

$$= 2(\frac{\sum_{i=1}^j p_i}{2})^2 + 2\gamma^2 \leq 2(\frac{\sum_{i=1}^j p_i}{2})^2 + 2(\frac{p_j}{2})^2.$$

As $\sum_{i=1}^{j-1} p_i \geq p_j$, then $\sum_{i=1}^j p_i = \sum_{i=1}^{j-1} p_i + p_j \geq 2p_j$, that is $p_j \leq \frac{\sum_{i=1}^j p_i}{2}$.

Hence, $C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT}) \leq 2(\frac{\sum_{i=1}^j p_i}{2})^2 + 2(\frac{\sum_{i=1}^j p_i}{4})^2 = \frac{5}{8}(\sum_{i=1}^j p_i)^2$. $\square$

Property 3 If $\sum_{i=1}^{j-1} p_i < p_j$, then $C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT}) \leq (p_j + \frac{\sum_{i=1}^{j-1} p_i}{2})^2 + (\frac{\sum_{i=1}^{j-1} p_i}{2})^2$.

Proof: From Lemma 1, we know that $C_{[j]}(S_{SPT}) \geq p_j$, namely $C_{[j]}(S_{SPT}) = p_j + \beta$ with $\beta \geq 0$. Correspondingly, from Lemma 3, we get

$$C_{[j-1]}(S_{SPT}) = \sum_{i=1}^j p_i - C_{[j]}(S_{SPT}) = \sum_{i=1}^{j-1} p_i - (C_{[j]}(S_{SPT}) - p_j) = \sum_{i=1}^{j-1} p_i - \beta$$

where $\beta := C_{[j]}(S_{SPT}) - p_j \leq \frac{\sum_{i=1}^{j-1} p_i}{2}$. Then

$$C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT}) \geq z = (p_j + \beta)^2 + (\sum_{i=1}^{j-1} p_i - \beta)^2.$$

But, as $p_j \geq \sum_{i=1}^{j-1} p_i$, we have that z is a function increasing in β and reaches then its maximum in the largest feasible value of β that is for $\beta = \frac{\sum_{i=1}^{j-1} p_i}{2}$. Hence, we have $C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT}) \leq (p_j + \frac{\sum_{i=1}^{j-1} p_i}{2})^2 + (\frac{\sum_{i=1}^{j-1} p_i}{2})^2$. $\square$

From the above properties, the approximation ratio of the SPT rule for the $P2||\sum C_j^2$ problem can be established as follows.

Proposition 1 *The approximation ratio of the SPT rule is bounded above by $\frac{\sqrt{5}+2}{\sqrt{5}+1} \approx 1.309$.*

Proof: Let assume w.l.o.g. that the number of jobs is even (in case of an odd number of jobs, it is sufficient to add a dummy job with processing time equal to 0). From Properties 1 and 2, we have that, if $\sum_{i=1}^{j-1} p_i \geq p_j$, then

$$\begin{aligned} C_{[j]}^2(S_{OPT}) + C_{[j-1]}^2(S_{OPT}) &\geq 2(\frac{\sum_{i=1}^{j-1} p_i}{2})^2 = \frac{1}{2}(\sum_{i=1}^j p_i)^2 \text{ and} \\ C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT}) &\leq \frac{5}{8}(\sum_{i=1}^j p_i)^2. \text{ Hence,} \\ \frac{C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT})}{C_{[j]}^2(S_{OPT}) + C_{[j-1]}^2(S_{OPT})} &\leq \frac{\frac{5}{8}(\sum_{i=1}^j p_i)^2}{\frac{1}{2}(\sum_{i=1}^j p_i)^2} = \frac{5}{4}. \end{aligned}$$

On the other hand, from Properties 1 and 3, we have that, if $\sum_{i=1}^{j-1} p_i < p_j$, then

$$\begin{aligned} C_{[j]}^2(S_{OPT}) + C_{[j-1]}^2(S_{OPT}) &\geq p_j^2 + (\sum_{i=1}^{j-1} p_i)^2 \text{ and} \\ C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT}) &\leq (p_j + \frac{\sum_{i=1}^{j-1} p_i}{2})^2 + (\frac{\sum_{i=1}^{j-1} p_i}{2})^2. \text{ Hence,} \\ \frac{C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT})}{C_{[j]}^2(S_{OPT}) + C_{[j-1]}^2(S_{OPT})} &\leq \tau = \frac{(p_j + \frac{\sum_{i=1}^{j-1} p_i}{2})^2 + (\frac{\sum_{i=1}^{j-1} p_i}{2})^2}{p_j^2 + (\sum_{i=1}^{j-1} p_i)^2}. \end{aligned}$$

Let have $y = \frac{\sum_{i=1}^{j-1} p_i}{2}$ and thus $\tau = \frac{(p_j + y)^2 + y^2}{p_j^2 + 4y^2}$. To compute a maximum on τ , we impose that the derivative of τ over p_j is equal to zero, that corresponds to impose, after easy algebraic calculations, $p_j^2 - 4y^2 - 2p_j y = 0$. The only positive root of this equation is then $p_j = y(1 + \sqrt{5})$. Correspondingly, function τ has a maximum for $p_j = (\sqrt{5} + 1) \frac{\sum_{i=1}^{j-1} p_i}{2}$ inducing $\frac{C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT})}{C_{[j]}^2(S_{OPT}) + C_{[j-1]}^2(S_{OPT})} \leq \frac{\sqrt{5}+2}{\sqrt{5}+1} \approx 1.309$.

Overall, we have $\frac{C_{[j]}^2(S_{SPT}) + C_{[j-1]}^2(S_{SPT})}{C_{[j]}^2(S_{OPT}) + C_{[j-1]}^2(S_{OPT})} \leq \max\{\frac{\sqrt{5}+2}{\sqrt{5}+1}, \frac{5}{4}\} = \frac{\sqrt{5}+2}{\sqrt{5}+1} \approx 1.309$. But then, summing up the above result on all pairs $j, j-1$ with $j = 2k$ for $k = 1, \dots, n/2$, we get the claimed result. $\square$

3 Lower bounds and final remarks

A non-tight (but close) lower bound on the approximation ratio for the $P2||\sum C_j^2$ problem, is given by the following example with 3 jobs with $p_1 = p_2 = 1$ and $p_3 = \sqrt{5} + 1$. An optimal schedule assigns jobs 1, 2 to machine M_1 and job 3 to machine M_2 with cost function value $11 + 2\sqrt{5}$, whilst the SPT rule assign jobs 1 to machine M_1 , job 2 to machine M_2 and job 3 arbitrarily to machine M_1 or M_2 with cost function value

$11 + 4\sqrt{5}$. Correspondingly the approximation ratio is $\frac{11+4\sqrt{5}}{11+2\sqrt{5}} \approx 1.289$.

A lower bound on the approximation ratio for the $Pm||\sum C_j^2$ problem, can be obtained by generalizing the example above to m machines, where we have $m+1$ jobs with jobs $1, \dots, m$ with processing time 1 and job $m+1$ with processing time $p_{m+1} = \sqrt{5} + 1$. An optimal schedule assigns job i to machine M_i with $i = 1, \dots, m-1$, job m arbitrarily to machine M_1 and job $m+1$ to machine M_m . Then, the cost function value is $m + 9 + 2\sqrt{5}$. On the other hand, the *SPT* rule assigns job i to machine M_i with $i = 1, \dots, m$ and job $m+1$ arbitrarily to machine M_1 with cost function value $m + 9 + 4\sqrt{5}$. Correspondingly the approximation ratio $\rho = \frac{m+9+4\sqrt{5}}{m+9+2\sqrt{5}}$ (for $m = 10$, for instance, we have $\rho \approx 1.191$).

The upper bound on the approximation ratio for 2 machines proposed in Proposition 1 seems to be generalizable to m machines. On the other hand, closing the gap between upper and lower bounds seem to be less intuitive.

References

1. T.C.E. Cheng, Z. Liu (2004), "Parallel machines scheduling to minimize the sum of quadratic completion times", *IIE Transactions* 36, 11-17.
2. W. Townsend (1978), "The single machine problem with quadratic penalty function of completion times: a branch-and-bound solution", *Management Science* 24, 530-534.

Impact of core precedences in a Cyclic RCPSP with temporal constraints

Z. Hanzalek · C. Hanen

Abstract In this paper, we introduce a new kind of constraints, called core precedence constraints, in a cyclic resource constrained scheduling problem with temporal constraints. We use them to model energy saving in a network scheduling problem. Then we study the impact of these new constraints on an efficient approach for cyclic RCPSP: the decomposed software pipelining approach. After discussion, we show that this approach can be used to prove that even without resource constraint, the existence of a periodic schedule is an $\mathcal{NP}$ -complete problem.

1 Introduction

Cyclic RCPSP with temporal constraints is a useful way to model scheduling problems occurring either in code generation for VLIW architectures [23][10] as well as in production systems [19][17]. In [14] the authors studied the scheduling problems induced by the IEEE 802.15.4/ZigBee network and they observed that the problem has a cyclic scheduling nature. This paper was first motivated by the use of tools and algorithms developed for solving cyclic RCPSP to solve this problem. However, our study lead us to introduce, within the cyclic RCPSP framework, the new concept of core precedences. Efficient periodic schedules in a cyclic RCPSP setting based on the concept of uniform precedences [23], are build by interleaving the iterations: in each period, the occurrence number of tasks is not necessarily the same. If this is necessary for our problem, temporal constraints and RCPSP resource settings are not sufficient to describe the whole problem. Indeed, energy efficiency constraints lead us to introduce precedence relations between tasks performed within a period of a periodic schedule (its core) regardless of their occurrence number.

We then discussed the extension of the usual approaches to cyclic RCPSP, and particularly the decomposed software pipelining approach described in [4]. This lead

Zdenek Hanzalek
Czech Technical University, 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

us, using the decomposed software pipelining tools, to prove that the combination of uniform precedences and core precedences leads to a $\mathcal{NP}$ -complete feasibility problem.

This paper is organized in 5 sections. Section 2 describes the IEEE 802.15.4/ZigBee messages scheduling problem and highlights its cyclic scheduling features. In section 3, we define formally our problem settings and notations, and introduce the core precedence constraints and the feasibility problem with and without resource constraints. Section 4 shows an ILP model of the problem. In section 5, we recall the main feature of the decomposed software pipelining approach, we discuss its application to our problem. Then in section 6, we use this approach to prove that the feasibility problem without resource constraints is $\mathcal{NP}$ -complete by a reduction from 3-SAT. The last section concludes the problem.

2 Motivating problem

In order to improve the functionality and efficiency of industrial monitoring and control systems, the industries are looking toward Wireless Sensor Networks (WSNs) to provide sensing and actuating in hazardous and previously hard-to-reach areas where very specialized and costly procedures must be adhered. The use of WSNs has many advantages compared to wired solutions. For example, WSNs may be installed for a fraction of the cost and time of an existing wired network. Since the wireless nodes are usually battery-powered, the network can be effectively used in environments where electricity is not available or some level of mobility is required (e.g. rotating parts of machines). Therefore, timeliness given by the control loop and energy efficiency given by the battery capacity are important requirements to be fulfilled in these systems.

The IEEE 802.15.4/ZigBee[5][21] standards are leading technologies for low-cost, low-power and low-rate WSNs. In the beacon-enabled variant, the cluster-tree network is set up, i.e. each node knows its parent and child nodes, and the paths of the data flows are determined by the tree topology. Time triggered Medium Access Control protocol is used to ensure collision avoidance.

We consider that all nodes may have sensing or/and actuating capabilities, therefore they can be sources or/and sinks of periodic data flows. The flows traverse different clusters on their routing paths from the source nodes to the sink nodes. The clusters may have collisions when they are in the neighborhood. Thus, the key problem is to find a periodic schedule which specifies when the clusters are active while avoiding possible inter-cluster collisions and meeting all flows' end-to-end deadlines.

In order to be energy efficient, the standard [21] specifies that each cluster is active only once during the period. This fact leads to a cyclic schedule, since end-to-end delay (i.e. time between the instant when a source sends the message and the instant when the sink receives this message) spans over several periods when there are the flows with opposite directions. Moreover, the flows are constrained by end-to-end deadlines and therefore not every instance has a feasible solution. The objective is usually to minimize the energy consumption of nodes by maximizing the schedule period (consequently maximizing time when the nodes stay in low-power mode).

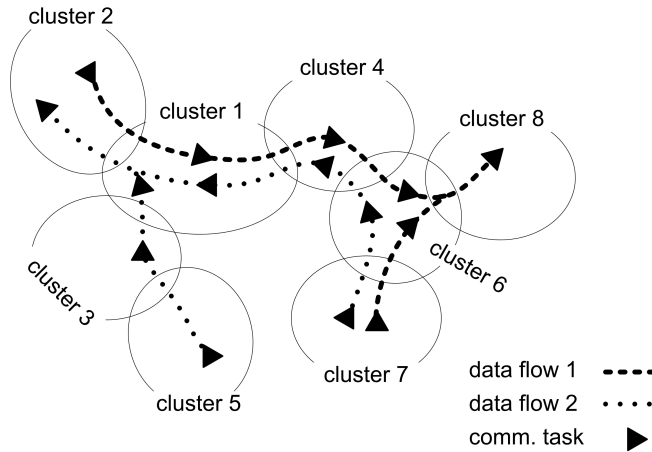


Fig. 1 Clusters with two data flows.

2.1 Resources

In the cluster-tree topology, the messages are forwarded from cluster to cluster until reaching the sink. The time behavior of each cluster is periodic and all clusters have the period of the same length. The period of each cluster is divided into two portions - active portion, during which the cluster-head enables the transmissions inside its cluster, and subsequent inactive portion. Length of the period is the same for all clusters. From the scheduling theory point of view, cluster j can be seen as a resource needed to perform communication among its members.

Due to the emission of significant radio energy, some the other clusters are blocked (i.e. they have to be in inactive portion) during the active portion of cluster j , otherwise the collision occurs. We define $\mathcal{M}$ as a set of couples of the tasks having a potential conflict. Consider two tasks T_i and T_j . The potential conflict between T_i and T_j is a couple $\{i, j\}$ derived from the topology of the system. For example task executed in cluster 1 have potential conflict with tasks executed in cluster 2, since these two clusters overlap.

Nevertheless the cyclic behavior, which we depict in this paper, occurs also in mono-resource case (when all clusters have one collision domain).

2.2 Data flows

The traffic is organized in the data flows. Particular task represents communication of given data flow in given cluster. Each data flow is represented by an acyclic oriented graph consisting of tasks and their precedence relations. See Fig. 1 where two data flows are communicated over eight clusters. The tasks are represented by small black triangles depicting also direction of the flow. End-to-end delay, given as a time between the instant when a source cluster i starts to communicate the message and the instant when the sink cluster j receives this message, is bounded by $\tilde{d}_{ij}$.

2.3 Cyclic nature of the problem

One execution of the flow (i.e. complete communication of messages from the source cluster/clusters to the sink cluster/clusters) is called an *occurrence*, and the notation f_i^q is used to denote occurrence q of the flow i . The flows are transmitted with the same period, therefore occurrence f_i^q is followed by occurrence f_i^{q+1} with the same time separation for all flows i and all iterations q . The key problem is to find a periodic schedule which specifies when the flows are communicated by clusters while avoiding possible inter-cluster collisions. The schedule is characterized not only by the start time of tasks within the period (further denoted core start time σ), but due to the cyclic nature of the problem it is also given by the shift of the period index (further denoted retiming α). The cyclic nature of the problem is given by the three aspects:

- In order to ensure minimal energy consumption, the cluster is active only once during the period [21]. Therefore, all the flows in the cluster are bound together and the cluster is inactive in the rest of the period.
- The end-to-end delay of the flow is constrained by deadline.
- The flows can have opposite directions, and therefore the best phasing (i.e. complete occurrence of the flow is executed in one period) of cluster activity in one direction is the worst phasing in the opposite direction (i.e. complete occurrence of the flow is executed in k periods, where k is the longest chain of clusters traversed by the two flows).

3 Problem Definition

In this section, we first introduce the cyclic RCPSP setting, and we show how it can be used to model clusters and data flows in the Zigbee scheduling problem. Then, we introduce core precedences and show how they can model the energy efficiency constraints in Zigbee. We then state and discuss the feasibility problems that are induced by these new constraints.

3.1 Cyclic RCPSP and uniform precedence constraints

Among the possible models for cyclic scheduling problems with temporal and resource constraints [13],[23],[17],[1], the Cyclic unitary RCPSP problem is one of the most useful, for which some approximated and ILP algorithms have been provided and experimented in practice [4],[10],[2]. In this section, we recall the main features of this model, then in the next section we show how it can be extended to handle the special constraints of our motivating example.

3.1.1 Statements and notations

In this model, the set of tasks $\mathcal{T}$ is supposed to be executed infinitely many times. For each task T_i , and each integer q , we denote by T_i^q the q^{th} occurrence of task T_i .

An infinite schedule s assigns to each occurrence T_i^q of task T_i a starting time s_i^q .

In the rest of the paper, we focus on periodic schedules, i.e. schedules such that each task is repeated every λ time unit: $\forall T_i \in \mathcal{T}, \quad s_i^q = s_i + (q-1)\lambda$, where s_i is the starting time of the first occurrence of T_i and λ is the period of the schedule.

The resource constraints are defined as follows:

1. m resource types. Availability of resource type $k \in \{1, \dots, m\}$ is denoted by R_k .
2. A set $\mathcal{T}$ of n tasks with integer processing time $\{p_i\}_{1 \leq i \leq n}$. To each task T_i and each type of resource k is associated an integer request r_{ik} , such that each occurrence of T_i uses r_{ik} units of type k resource during its execution.
3. If s is a schedule then for any time unit t , and any resource k , the number of resource requirements made by the tasks T_i^q such that $s_i^q \leq t < s_i^q + p_i$ is not greater than R_k .
4. The problem is called unitary if $r_{ik} \in \{0, 1\}$, which occurs for our example.

Now the usual model for cyclic precedence constraints is called "uniform precedences". Such constraints are defined by a bivalued graph $G_U = (\mathcal{T}, \mathcal{E}_U)$ with integer valuations l and h called respectively length and height of its arcs. For an arc $a = (T_i, T_j)$ we denote by l_{ij} its length and by h_{ij} its height.

An arc $a = (T_i, T_j)$ of G_U induces an infinite number of precedence constraints on any schedule s as follows: for any iteration q , T_i^q precedes $T_j^{q+h(a)}$, i.e.

$$s_j^{q+h_{ij}} - s_i^q \geq l_{ij}$$

Notice that for a periodic schedule, a uniform constraint induces a simple linear inequality [13] :

$$s_j - s_i \geq l_{ij} - \lambda h_{ij}$$

For a path μ of G_U , we denote by $L(\mu)$ the sum of length of its arcs, and by $H(\mu)$ the sum of heights of its arcs. In the rest of the paper, we shall oftenly discuss results depending on the fact that lengths, or heights, or both are nonnegative. We first show that nonnegative heights and lengths might be a convenient assumption for our Zigbee problem.

3.1.2 Modeling data-flows and clusters resource constraints in Zigbee

This model may be convenient to represent the precedence relations within each data flow of our motivating example. Indeed, if two tasks of data-flow communicate (from a cluster to a neighbor cluster), this communication may induce a precedence constraint with delay between occurrences of these two tasks belonging to the same iteration, and thus induce a uniform precedence constraint with height equal to 0.

On the other side, the end-to-end delay $\tilde{d}_{ij}$ mentioned in section 1.2. between the data emission of the source cluster (let us call it T_i) of an occurrence of a data-flow, and the reception of the data by the sink cluster (called T_j) can be modeled by two ways:

As it concerns a single data flow, it can result in a uniform constraint from T_j to T_i with length $p_j - \tilde{d}_{ij}$ and height 0.

But in fact if the period λ is given, or if a sufficiently small interval is given for periods, then as we know that an occurrence of a data flow will span over several periods, one way to limit its end-to-end delay is to limit the number of periods across which the data flow schedule will span. Thus in our example, to define an arc from T_j to T_i with fixed height. For example a height 3 implies that in any periodic schedule, the difference $s_i - s_j \geq p_j - \lambda * 3$, so that $s_j + p_j \leq s_i + 3\lambda$.

Notice that in both models, the heights of the arcs of G_U are nonnegative, and that in the second case, the lengths of the arcs are also nonnegative.

Now considering that a cluster cannot execute more than one task at the same time can be modeled by a resource with availability 1. Each data-flow task executed by the cluster has a resource requirement of 1 for that resource.

3.1.3 Core of a periodic schedule

Let s be a periodic schedule of period λ . It is useful to consider another representation of s , that will be used in the next section.

For a task T_i , we decompose s_i , the starting time of its first occurrence modulo the period λ : We call retiming of T_i , and denote by α_i the greatest integer such that there exist $\sigma_i \in [0, \lambda)$ with $s_i = \sigma_i + \alpha_i \lambda$.

$(\sigma_i)_{i \in \mathcal{T}}$ is called the core of the periodic schedule. Indeed, we can observe that in any feasible periodic schedule with period λ , if we consider an interval $[q\lambda, (q+1)\lambda)$ of the steady state (for large enough q), then σ_i is the starting time of the occurrence of T_i in this interval (from time origin $q\lambda$).

Thus, σ_i is an acyclic schedule of tasks that fulfils the resource constraints. The retiming α_i models the span of the schedule of an iteration over different periods.

Notice that a uniform precedence constraint (T_i, T_j) induces the following inequality:

$$\sigma_j - \sigma_i \geq l_{ij} - \lambda(h_{ij} + \alpha_j - \alpha_i) \quad (1)$$

3.1.4 State of the art and bounds

Most of the results on cyclic scheduling problems aim at minimizing the period of a feasible schedule. The related literature also refers to "modulo scheduling" [22] and "software pipelining" approaches [1][16]. It is well known that if no resource constraints are considered, given a uniform graph G_U , the feasibility problem as well as the period minimization problem can be solved in polynomial time, even if integer lengths and heights are considered. According to [23][20]:

Lemma 1 *If G_U is feasible, then any circuit c such that $H(c) = 0$ satisfies $L(c) \leq 0$. Moreover if heights are nonnegative it is a necessary and sufficient feasibility condition.*

If this condition is met, we can define the two following Bounds:

$$B_1 = \max_{c \text{ circuit of } G_U, H(c) > 0} \frac{L(c)}{H(c)} \quad B_2 = \min_{c \text{ circuit of } G_U, H(c) < 0} \frac{L(c)}{H(c)} \quad (2)$$

Following [23][20], the two bounds can be computed in polynomial time, for example with the parametric algorithms given in [17], or other algorithms experimented in [7] and we get:

Lemma 2 *G_U is feasible if and only if $B_1 \leq B_2$ and $\forall \lambda \in [B_1, B_2]$, there exist a periodic schedule with period λ satisfying the constraints of G_U , that can be found in polynomial time.*

Notice that if nonnegative heights are assumed, which is the case for our Zigbee model, then B_2 vanishes and B_1 , the value of the critical circuit, is the minimal period of a periodic schedule.

Now, finding a modulo schedule of minimal period λ is known to be $\mathcal{NP}$ -hard when resources are limited [13].

On the other hand, a bound due to resource constraints only can be defined by

$$\lambda^{res} = \max_{1 \leq k \leq m} \frac{\sum_{T_i \in \mathcal{T}} p_i r_{ik}}{R_k}.$$

That is the minimum such that the renewable resources are not overloaded and it can be easily proven that $\lambda^{opt} \geq \lambda^{res}$.

Several approaches have been investigated to solve the cyclic RCPSP problem with nonnegative heights and lengths. Integer Linear Programming [2], greedy iterative heuristics [16][15][22][18][24], mixed approaches [9][8][3], and the so called decomposed software pipelining approach [4][6][25][12] which will be detailed in the next section. In this paper we focus on this approach to discuss the feasibility of our new problem, and to propose some heuristics.

3.2 Core precedence constraints

Now, in our example, the constraints that bind the tasks in a cluster in order to insure minimal energy consumption cannot be modeled by cyclic RCPSP constraints. Thus we introduce a new type of constraints, called core precedences, aimed at modeling these constraints. Consider an instance of the cyclic RCPSP problem described in the previous section. A core precedence constraint links the core execution times of two tasks T_i and T_j with a time lag denoted by $c_{ij} \in \mathbb{Z}$

$$\sigma_j - \sigma_i \geq c_{ij} \quad (3)$$

We consider a graph $G_C = (\mathcal{T}, E_C)$ of such core constraints in addition with an instance of the RCPSP problem.

Notice that a core precedence links the execution times of tasks occurrences in any period interval of the steady state of a periodic schedule, regardless of the iteration number of the considered occurrences.

So, it shares this indifference with usual resource constraints, whereas it shares the notion of precedence with uniform constraints. Such constraints might also result from partial choices with respect to usual resource constraints.

3.2.1 Modeling energy efficiency in Zigbee

In our motivating example, all tasks of the different data-flows that use a given cluster are supposed to be grouped during each period, so as to minimize the energy consumption, regardless of the iteration number of the concerned data-flow.

This can be modeled by a cluster dummy task T_i representing the starting of the cluster activity with processing time 0. For each task T_j of a data flow passing through the cluster, we can define two core precedences :

- (T_i, T_j) with core value 0 that model the fact that the data-flow task always starts after the dummy task in any period.
- (T_j, T_i) with core value $-x$ where x is the maximum allowed activation interval of the cluster in a period.

in this way we ensure that all tasks performed by the cluster are executed within x time units and that the cluster is inactive the rest of the period.

3.2.2 Feasibility of Core constraints

Introducing core constraints in a RCPSP setting leads to the following questions:

- If the problem without core precedences is feasible, then does it exist a periodic schedule?
- For a given period λ does it exist a feasible periodic schedule with period λ ?
- How can the problem of the minimization of λ be expressed and solved?

To the first question, the answer is no, according to the following example, which assumes no resource constraints, only nonnegative core and uniform precedences:

- G_U is a simple circuit of 3 unit processing time tasks T_1, T_2, T_3 , with $l_{i,i+1} = 1, h_{i,i+1} = 0, l_{3,1} = 1, h_{3,1} = 1$
- G_C is composed of two arcs: (T_2, T_1) with $c_{2,1} = 1$ and (T_1, T_3) with $c_{1,3} = 1$.

Clearly, in any feasible core, $\sigma_2 < \sigma_1 < \sigma_3$. According to inequality 1, considering a uniform arc (T_i, T_j) : $\sigma_j - \sigma_i \geq l_{ij} - \lambda(h_{ij} + \alpha_j - \alpha_i)$.

This will imply that

$$\alpha_j - \alpha_i \geq -h_{ij} + \left\lceil \frac{l_{ij} + \sigma_i - \sigma_j}{\lambda} \right\rceil$$

Applying this inequality to arc (T_1, T_2) leads to $\alpha_2 \geq \alpha_1$ since $\sigma_1 - \sigma_2 < 0$.

Similarly, for arc (T_2, T_3) we get $\alpha_3 \geq \alpha_2$. Consider now arc (T_3, T_1) : $\alpha_1 - \alpha_3 \geq -1 + \left\lceil \frac{1 + \sigma_3 - \sigma_1}{\lambda} \right\rceil$. As $\sigma_3 > \sigma_1$, $\left\lceil \frac{1 + \sigma_3 - \sigma_1}{\lambda} \right\rceil \geq 1$, so that $\alpha_1 \geq \alpha_3$ and then all α_i are equal.

But then, the uniform arc (T_1, T_2) and the core arc (T_2, T_1) are not compatible.

This result lead us to define the following problems:

- **UCF**: uniform and core feasibility. Given a uniform graph G_U and a core graph G_C does it exist a feasible periodic schedule satisfying both?
- **UCF_fixed_period**: Given a uniform graph G_U and a core graph G_C does it exist a feasible periodic schedule with a given period λ satisfying both?
- **RCUCF**: resource constrained uniform and core feasibility. Given a cyclic resource constrained scheduling problem with core constraints, does it exist a feasible periodic schedule?
- **RCUCF_fixed_period**: Given a cyclic resource constrained scheduling problem with core constraints, does it exist a feasible periodic schedule with a given period λ ?
- **RCUCF_opt**: Given a cyclic resource constrained scheduling problem with core constraints, what is the minimum λ for which a periodic schedule exists?

Notice that as finding a periodic schedule with a given period λ (or minimizing) given a cyclic RCPSP problem setting is a NP-hard problem, the problems **RCUCF_fixed_period** and **RCUCF_opt** are still NP-hard since core precedence constraints are added.

4 ILP model for RCUCF_fixed_period

Several ILP models have been proposed for cyclic RCPSP problems [10] assuming nonnegative lengths and heights. According to the authors of [2], the more efficient model

is the generalisation of time indexed Eichenberger model. Notice that this model as well as the other model proposed in [14] can be easily extended to handle core precedence constraints and arbitrary lengths and heights.

Eichenberger and Davidson [11] initially describe a model with unit processing time tasks for the feasibility of a periodic schedule with period λ . This model can be extended to handle integer processing time [10]. It uses as variables the core variables σ_i , the integer retiming variables α_i and some time indexed binary variables related to the core:

y_i^τ such that, $y_i^\tau = 1$ if only if $\sigma_i = \tau$

$$\sigma_i = \sum_{\tau=0}^{\lambda-1} \tau y_i^\tau \quad (4)$$

$$\sum_{\tau=0}^{\lambda-1} y_i^\tau = 1, \quad \forall i \in \{1, \dots, n\} \quad (5)$$

$$\sigma_j - \sigma_i \geq l_{ij} - \lambda(h_{ij} + \alpha_j - \alpha_i) \quad \forall (T_i, T_j) \in E_U \quad (6)$$

$$\sum_{i=1}^n \sum_{\gamma=\max(0, \tau\gamma-p_i+1)}^{\gamma} y_i^\tau r_{ik} \leq R_k, \quad \forall k \in \{1, \dots, m\}, \forall \gamma \in \{0, \dots, \lambda-1\} \quad (7)$$

$$y_i^\tau \in \{0, 1\} \quad \forall i \in \{1, \dots, n\}, \forall \tau \in \{0, \dots, \lambda-1\} \quad (8)$$

$$\alpha_i \in \{0, \dots, K-1\} \quad \forall i \in \{1, \dots, n\} \quad (9)$$

Constraints (6) are the precedence constraints. Constraints (5) state that each generic operation has to be started exactly once in the period or, equivalently, that the reminder of the division of σ_i by λ lies in $\{0, \dots, \lambda-1\}$. With this decomposition, the set of operations such that $y_i^\tau = 1$ directly gives resource constraints (7).

Core precedence can also be included in the model, following equation (3)

$$\sigma_j - \sigma_i \geq c_{ij} \quad \forall (T_i, T_j) \in E_C$$

So, RCUCF.fixed_period as well as UCF.fixed_period can be modeled as an ILP. Notice that if we try to solve RCUCF_opt, the constraint (6) is not anymore linear, since the period λ multiplies the retiming variable. So to solve this problem, a binary search on λ should be made, with at each step an ILP solving for fixed λ .

As ILP may fail to find a solution in a reasonable time for real instance, in the rest of the paper we focus on some theoretical tools that might be used to find heuristics for solving these problems.

5 Decomposed software pipelining for cyclic RCPSP with core constraints

The decomposed software pipelining approach has been proven very efficient in practice to solve cyclic RCPSP problems with nonnegative lengths and heights. It is analyzed in [4], and [3].

In this section we recall the main features of this approach, and then we discuss its extension to handle core constraints. Then, we show that DSP can be used as a framework to prove the $\mathcal{NP}$ -completeness of the feasibility problem without resource constraints.

5.1 Decomposed software pipelining

Assume that we get a RCPSP instance, with a uniform graph G_U with nonnegative lengths and heights. This approach consists in decomposing the construction of a feasible schedule into three successive steps:

1. Find a feasible retiming $(\alpha_i)_{i \in \mathcal{T}}$, i.e. a retiming such that for any arc $(T_i, T_j) \in E_U$, $h_{ij} + \alpha_j - \alpha_i \geq 0$.
2. Build with any algorithm a schedule σ of the acyclic graph G_U^α that fulfils resource constraints
3. Build the periodic schedule whose period is the makespan of σ , whose core is σ and whose retiming is α .

Regarding the second step, the graph G_U^α is defined by removing from G_U the arcs for which $h_{ij} + \alpha_j - \alpha_i > 0$, keeping their length, and adding two dummy nodes T_{start} source of the graph, and T_{end} sink of the graph with null processing times, with appropriate lengths of their incident arcs.

It has been proved in [4] that if:

1. A retiming minimizing the length of the longest path of G_U^α or based on a schedule build by assuming infinitely many resources is used at step 1;
2. A list schedule is used at step 2 and if unitary resource demands are assumed ($r_{ik} \in \{0, 1\}$);

then DSP provides a polynomial time approximation algorithm with a bounded worst case ratio.

5.2 DSP and core constraints

If we consider now our problem with core constraints and any integer lengths, this approach might be extended, but several problems have to be faced.

5.2.1 Nonnegative lengths and heights

Let us first assume nonnegative lengths and heights of the arcs of G_U . Notice that this holds for our Zigbee problem if we assume that the end-to-end delay of data-flows are modeled as suggested in section 3.2.1 by arcs with limited height. Then, the definition of feasible retiming still holds. The following algorithm can then provide a feasible schedule:

1. Find a feasible retiming $(\alpha_i)_{i \in \mathcal{T}}$, i.e. a retiming such that for any arc $(T_i, T_j) \in E_U$, $h_{ij} + \alpha_j - \alpha_i \geq 0$.
2. Build with any algorithm a schedule σ of the joint graph $\mathcal{H}^\alpha$ combining the arcs of G_U^α and the arcs of the core constraints G_C that fulfils resource constraints.
3. Build the periodic schedule whose period is the makespan of σ , whose core is σ and whose retiming is α .

Unfortunately, as mentioned in the previous section, the core constraints may contradict the uniform constraints, so that, it may happen that $\mathcal{H}^\alpha$ is not anymore acyclic. However, if we are able to find a retiming so that $\mathcal{H}^\alpha$ is acyclic then this approach yields a feasible periodic schedule.

Notice that if we can solve the underlying UCF problem (considering only G_U and G_C constraints), then if we provide a periodic schedule for this problem, the associated retiming will have the property that $\mathcal{H}^\alpha$ is acyclic so this algorithm would provide a feasible periodic schedule. Hence in order to use DSP approach, a further study of UCF should be made.

5.2.2 Integer lengths and nonnegative heights

If we allow some negative lengths, then the feasibility condition of a retiming is no longer that for any arc $h_{ij} + \alpha_j - \alpha_i \geq 0$. There might be arcs for which this value is negative. Moreover we cannot consider that for a given retiming α , $\mathcal{H}^\alpha$ is acyclic. However the retiming notion can be used to derive some new ideas:

Consider now general length of the arcs of G_U . The feasibility condition of G_U is then that for any circuit c , if $H(c) = 0$ then $L(c) < 0$.

Let us assume that we are given a periodic schedule s^∞, λ^∞ satisfying G_U and G_C , and thus a solution to the UCF problem.

Then, if we define for any task i the core starting time σ_i^∞ and the retiming α_i^∞ of the task i , let us define $\mathcal{H}^{\alpha^\infty}$ the graph composed with *all* arcs of G_U and arcs of G_C . Let us define the length l'_{ij} and the height h'_{ij} of an arc of $\mathcal{H}^{\alpha^\infty}$ as follows:

$$\forall (T_i, T_j) \in E_U, \quad l'_{ij} = l_{ij} \quad (10)$$

$$\forall (T_i, T_j) \in E_C, \quad l'_{ij} = c_{ij} \quad (11)$$

$$\forall (T_i, T_j) \in E_U, \quad h'_{ij} = h_{ij} + \alpha_j^\infty - \alpha_i^\infty \quad (12)$$

$$\forall (T_i, T_j) \in E_C, \quad h'_{ij} = 0 \quad (13)$$

Let us denote by $L'(\mu)$ (resp. $H'(\mu)$) the sum of lengths (resp. heights) of the arcs of the path μ . Notice that unlike the previous case, the height $H'(\mu)$ of a path might be negative: indeed, we get the following property:

Lemma 3 *If c is a circuit of $\mathcal{H}^{\alpha^\infty}$, then $H'(c) \leq 0$ implies that $L'(c) < 0$*

Proof. let us sum-up the inequalities satisfied by σ^∞ along the circuit c . We get:

$$0 \geq L'(c) - \lambda^\infty H'(c)$$

we can define the two following Bounds:

$$B_1 = \max_{c \text{ circuit of } \mathcal{H}^{\alpha^\infty}, H'(c) > 0} \frac{L'(c)}{H'(c)} \quad B_2 = \min_{c \text{ circuit of } \mathcal{H}^{\alpha^\infty}, H'(c) < 0} \frac{L'(c)}{H'(c)} \quad (14)$$

Following [23, 20], the two bounds can be computed in polynomial time, and we get:

Lemma 4 $\forall \lambda \in [B_1, B_2]$, *there exist a periodic schedule with period λ satisfying the constraints of G_U and G_C , that can be found in polynomial time.*

Proof. In order to build such a periodic schedule, we first build a schedule σ on the precedence graph $\mathcal{H}^{\alpha^\infty}$, with the following values: $\forall (T_i, T_j), \quad v_{ij} = l'_{ij} - \lambda h'_{ij}$. Notice that according to the definitions of the two bounds, every circuit of $\mathcal{H}^{\alpha^\infty}$ has a nonpositive value.

Then, we build σ so that for each arc (T_i, T_j) of $\mathcal{H}^{\alpha^\infty}$,

$$\sigma_j - \sigma_i \geq v_{ij}$$

Combining σ with the retiming α^∞ will give a feasible periodic schedule of period λ .

Now, if we consider resource constraints, assuming that α^∞ and σ^∞ are given, we can use the DSP approach as follows:

1. For a given $\lambda \in [B_1, B_2]$, try to find a feasible schedule σ of $\mathcal{H}^{\alpha^\infty}$ with valuation v as above that satisfies the resource constraints, using any heuristic.
2. If such a schedule is found, build a periodic schedule based upon σ and α^∞ . Otherwise, try another λ

6 Complexity of UCF

As shown in the previous section, UCF is a key problem, and if we could get solutions of UCF, the resolution of all problems with resource constraints could be made easier. However, we prove in this section that whereas it has no resource constraints, the UCF problem is $\mathcal{NP}$ -complete.

Let us consider two graphs G_U and G_C , and let us assume that the lengths of the arcs are unitary.

In this case the feasibility problem without resource constraints can be expressed as follows:

- Does it exist a feasible retiming α such that $\mathcal{H}^\alpha$ is acyclic?

Remark 1 Let α be a feasible retiming.

Then for any circuit c of G_U , there are at most $H(c)$ arcs (T_i, T_j) of c such that $h_{ij} + \alpha(j) - \alpha(i) > 0$. Thus, at most $H(c)$ arcs of c have been removed while defining G_U^α from G_U .

Let us consider an instance of the 3-SAT problem. We are given n binary variables $x_1, \dots, x_n$ and m clauses $C_1, \dots, C_m$. Each clause is defined by 3 literals. A literal is either an x_i or its complement $\bar{x}_i$.

The decision problem 3-SAT can be formulated as follows:

- Does it exist binary values of the variables such that in each clause, at least one literal is true?

This problem can be illustrated by the following example with 4 variables and 3 clauses:

$$C_1 = \{x_1, \bar{x}_2, x_4\}, \quad C_2 = \{x_2, \bar{x}_3, x_4\}, \quad C_3 = \{\bar{x}_1, \bar{x}_3, \bar{x}_4\}$$

For this example, $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0$ provides satisfiability of this system.

Let us associate with a 3-SAT instance an instance of our feasibility problem. Let us define, for each variable x_i , two tasks denoted by x_i and $\bar{x}_i$ with an arc of G_U from x_i to $\bar{x}_i$ with length 1 and height 0, and an arc from $\bar{x}_i$ to x_i with length 1 and height 1. These arcs are called "variable arcs".

Now let us define for each clause C_j four tasks $c_j^0, c_j^1, c_j^2, c_j^3$. Let us add in G_U the following arcs with length 1 representing the literals : $(c_j^0, c_j^1), (c_j^1, c_j^2), (c_j^2, c_j^3)$ with height 0 and (c_j^3, c_j^0) with height 2. These arcs are called "clause arcs".

Notice that according to remark 1, in any feasible retiming α , exactly one of the variable arcs related to x_i will be removed to get G_U^α , while at most two of the clause arcs related to clause C_j will be removed to get G_U^α .

Let us assume without loss of generality that the literals are ordered with respect to the index number of the associated variables (so if x_i or $\bar{x}_i$ is the first literal (resp. second), x_j or $\bar{x}_j$ the second (resp. third) literal, then $i < j$). Let us now define the core arcs as follows: assume that x_i is the literal $k \in \{1, 2, 3\}$ of clause C_j . Then (x_i, c_j^{k-1}) and $(c_j^k, \bar{x}_i)$ are arcs of G_C . If $\bar{x}_i$ is the literal k in clause C_j , then $(\bar{x}_i, c_j^{k-1})$ and (c_j^k, x_i) are arcs of G_C .

Notice that c_j^0 has only one incoming core arc and no outgoing core arc, whereas c_j^3 has only one outgoing core arc and no incoming core arc. c_j^1, c_j^2 both have one incoming and one outgoing core arcs. Figure 2 shows the graph associated to the example.

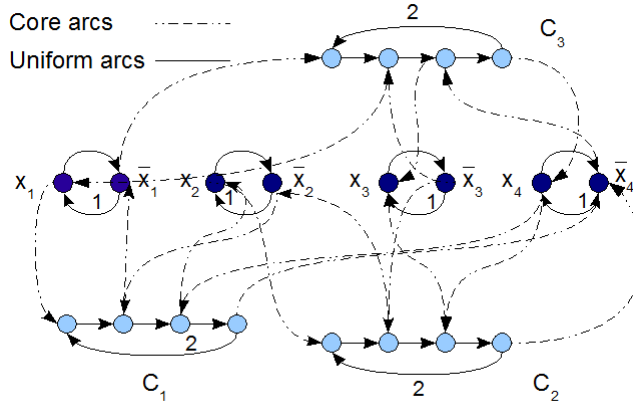


Fig. 2 A polynomial reduction

Lemma 5 *If there is a feasible retiming α such that $\mathcal{H}^\alpha$ is acyclic, then the 3-SAT instance is satisfiable.*

Proof. For each variable x_i , let us set $x_i = 1$ if the arc $(x_i, \bar{x}_i)$ is in $\mathcal{H}^\alpha$, $x_i = 0$ otherwise.

We know that for each clause C_j there is at least one arc (c_j^{k-1}, c_j^k) which remains in $\mathcal{H}^\alpha$.

Let us assume that the k^{th} literal of clause C_j is x_i . Then, as we have the core arc (x_i, c_j^{k-1}) , the clause arc (c_j^{k-1}, c_j^k) and the core arc $(c_j^k, \bar{x}_i)$, if $\mathcal{H}^\alpha$ has no circuit, then the variable arc $(\bar{x}_i, x_i)$ cannot belong to $\mathcal{H}^\alpha$. Therefore $(x_i, \bar{x}_i)$ is in $\mathcal{H}^\alpha$, and $x_i = 1$, so that clause C_j is true.

Let us now assume that k^{th} literal of clause C_j is $\bar{x}_i$. Then as we have the core arc $(\bar{x}_i, c_j^{k-1})$, the remaining arc (c_j^{k-1}, c_j^k) and the core arc (c_j^k, x_i) , the variable arc

$(x_i, \bar{x}_i)$ cannot belong to $\mathcal{H}^\alpha$, and therefore $(\bar{x}_i, x_i)$ is in $\mathcal{H}^\alpha$ and $x_i = 0$, so that the clause C_j is true.

Let us now consider the reciprocal of this lemma:

Lemma 6 *If the 3 – SAT instance is satisfiable then there exist a retiming α such that $\mathcal{H}^\alpha$ is acyclic.*

Proof. Assume that in the satisfiability assignment $x_i = 1$. Then we set $\alpha_{x_i} = \alpha_{\bar{x}_i} = 0$, so that the variable arc $(x_i, \bar{x}_i)$ remains in $\mathcal{H}^\alpha$ whereas the other vanishes. If $x_i = 0$, then we set $\alpha_{x_i} = 0, \alpha_{\bar{x}_i} = 1$, so that the variable arc $(\bar{x}_i, x_i)$ remains in $\mathcal{H}^\alpha$.

Consider now a clause C_j , and its first true literal $k \in \{1, 2, 3\}$. We define the retiming of tasks of clause C_j as follows:

- If $k = 1$ then $\alpha_{c_j^0} = \alpha_{c_j^1} = 0, \alpha_{c_j^2} = 1, \alpha_{c_j^3} = 2$, so that in $\mathcal{H}^\alpha$ the only remaining clause arcs for C_j are (c_j^0, c_j^1) and (c_j^3, c_j^0) .
- if $k = 2$ then $\alpha_{c_j^0} = 0, \alpha_{c_j^1} = \alpha_{c_j^2} = 1, \alpha_{c_j^3} = 2$, so that in $\mathcal{H}^\alpha$ the only remaining clause arcs for C_j are (c_j^1, c_j^2) and (c_j^3, c_j^0) .
- if $k = 3$ then $\alpha_{c_j^0} = 0, \alpha_{c_j^1} = 1, \alpha_{c_j^2} = \alpha_{c_j^3} = 2$ so that in $\mathcal{H}^\alpha$ the only remaining clause arcs for C_j are (c_j^2, c_j^3) and (c_j^3, c_j^0) .

Figure 3 shows the graph $\mathcal{H}^\alpha$ for the assignment $x_1 = 1, x_2 = x_3 = x_4 = 0$.

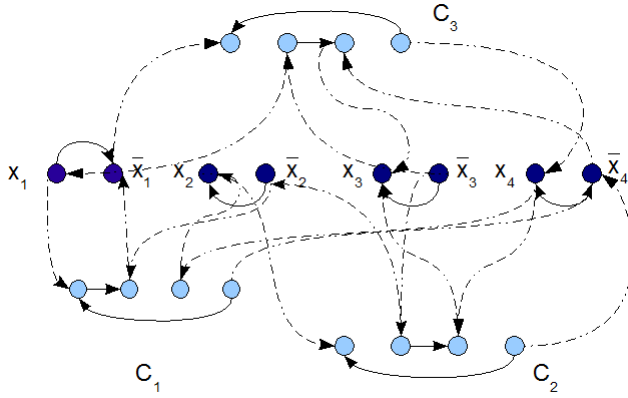


Fig. 3 A graph $\mathcal{H}^\alpha$

Let us assume that there is a circuit μ in $\mathcal{H}^\alpha$.

We first prove that there cannot be a clause C_j for which the arc (c_j^3, c_j^0) belongs to μ . Indeed, the only possible incoming arc to c_j^3 is the arc (c_j^2, c_j^3) . The only possible outgoing arc from c_j^0 is (c_j^0, c_j^1) . As at least one of them is not in $\mathcal{H}^\alpha$, this arc cannot belong to the circuit.

Let us now prove that if (c_j^{k-1}, c_j^k) belongs to μ then there is another circuit which does not pass through this arc. Notice that c_j^{k-1} has only one incoming arc that may belong to μ (from a variable node x_i (resp. $\bar{x}_i$)) and c_j^k has only one outgoing arc that may belong to μ (to the complemented variable node $\bar{x}_i$ (resp. x_i)), we can

define the part μ' of the circuit from $\bar{x}_i$ (resp. x_i) to x_i (resp. $\bar{x}_i$). By construction, the corresponding literal is true in the satisfiability problem, so the remaining arc $(x_i, \bar{x}_i)$ (resp. $(\bar{x}_i, x_i)$) added to μ' defines a new circuit. Hence if there is a circuit in $\mathcal{H}^\alpha$, there is a circuit that does not use any of the clause arcs.

Let us now consider the variable node of μ with the maximum index. Without loss of generality, assume that it is node x_i (resp. $\bar{x}_i$). The predecessor of x_i on the circuit should be a clause node c_j^k . The predecessor or c_j^k on μ is then a variable node, say $x_{i'}$ (resp. $\bar{x}_{i'}$). Thus, according to the definition of arcs, $\bar{x}_i$ (resp. x_i) should be the k^{th} literal of clause C_j , whereas $x_{i'}$ (resp. $\bar{x}_{i'}$) should be the $(k+1)^{th}$ literal of the same clause. But as we assumed that the literals are ordered by increasing index, $i' > i$, which contradicts the fact that x_i (resp. $\bar{x}_i$) has a maximum index on μ .

From this we deduce our complexity result:

Theorem 1 *The feasibility of a cyclic problem with core precedences and without resource constraint is $\mathcal{NP}$ -complete.*

Proof. given a retiming α , a simple depth first search algorithm can check the existence of a circuit in $\mathcal{H}^\alpha$ in polynomial time, so the feasibility problem is in $\mathcal{NP}$. Using the previous transformation, and lemma 5 and 6 we get our result.

7 Conclusion

In this paper, we defined, in the cyclic RCPSP framework, a new kind of constraints called core precedence constraints. We shown how such constraints can be used to model energy saving in Zigbee. Then we discussed about the interest of the DSP approach for finding feasible solutions or minimizing the period. And finally we use this approach to prove that even without resource constraints, the introduction of core precedence constraints makes the feasibility of a schedule a $\mathcal{NP}$ -complete problem. This paper is a first insight for this new kind of constraints. New ideas and algorithms should be further investigated in order to solve efficiently the problems with and without resource constraints.

References

1. Vicki H. Allan, Reese B. Jones, Randall M. Lee, and Stephen J. Allan. Software pipelining. *ACM Comput. Surv.*, 27(3):367–432, 1995.
2. M. Ayala and C. Artigues. On integer linear programming formulations for the resource-constrained modulo scheduling problem. *LAAS report 10393*, 2010.
3. Maria Ayala, Abir Benabid, Christian Artigues, and Claire Hanen. The resource-constrained modulo scheduling problem: an experimental study. *Technical report, submitted to Parallel Computing*, January 2011.
4. A. Benabid and C. Hanen. Decomposed software pipelining for cyclic unitary rcpsp with precedence delays. *Journal of Scheduling*, 2011.
5. 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.
6. Pierre-Yves Calland, Alain Darté, and Yves Robert. Circuit retiming applied to decomposed software pipelining. *IEEE Trans. Parallel Distrib. Syst.*, 9(1):24–35, 1998.
7. Ali Dasdan, Sandy Irani, and Rajesh K. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. In *Design Automation Conference*, pages 37–42, 1999.
8. B. Dupont de Dinechin. From machine scheduling to vliw instruction scheduling. *ST Journal of Research*, 1(2), 2004.

9. B. Dupont de Dinechin. Time-indexed formulations and a large neighborhood search for the resource-constrained modulo scheduling problem. In P. Baptiste, G. Kendall, A. Munier-Kordon, and F. Sourd, editors, *3rd Multidisciplinary International Scheduling conference: Theory and Applications*. 2007.
10. B. Dupont de Dinechin, C. Artigues, and S. Azem. Resource constrained modulo scheduling. In C. Artigues, S. Demasse, and E. Neron, editors, *Resource-Constrained Project Scheduling: models, algorithms, extensions and applications*, Control systems, robotics and manufacturing series, pages 267–277. ISTE and John Wiley, London, 2008.
11. A. Eichenberger and E.S. Davidson. Efficient formulation for optimal modulo schedulers. *SIGPLAN - PLDI'97*, 1997.
12. Franco Gasperoni and Uwe Schwiegelshohn. Generating close to optimum loop schedules on parallel processors. *Parallel Processing Letters*, 4:391–403, 1994.
13. 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.
14. 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.
15. Richard A. Huff. Lifetime-sensitive modulo scheduling. In *In Proc. of the ACM SIGPLAN '93 Conf. on Programming Language Design and Implementation*, pages 258–267, 1993.
16. M. Lam. Software pipelining: an effective scheduling technique for vliw machines. *SIGPLAN Not.*, 23(7):318–328, 1988.
17. Eugene Levner, Vladimir Kats, and David Alcaide Lopez de Pablo. Cyclic scheduling in robotic cells: An extension of basic models in machine scheduling theory. In Eugene Levner, editor, *Multiprocessor Scheduling: Theory and Applications*, pages 001–020. I-Tech Education and Publishing, Vienna Austria, 2007.
18. Josep Llosa, Antonio González, Eduard Ayguadé, and Mateo Valero. Swing modulo scheduling: A lifetime-sensitive approach. pages 80–86, 1996.
19. J. M. and Proth X. Xie. *Modélisation, analyse et optimisation des systèmes à fonctionnement cyclique*. Masson, 1995.
20. Alix MunierKordon. A graph-based analysis of the cyclic scheduling problem with time constraints: schedulability and periodicity of the earliest schedule. *Journal of Scheduling*, pages 1–15, 2010.
21. ZigBee Standards Org. Zigbee specification, 2006.
22. B. Ramakrishna Rau. Iterative modulo scheduling: an algorithm for software pipelining loops. In *MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture*, pages 63–74, New York, NY, USA, 1994. ACM.
23. Yves Robert and Frederic Vivien. *Introduction to Scheduling*. CRC Press, Inc., Boca Raton, FL, USA, 2009.
24. M. Smelyanskiy, S. Mahlke, and E.S. Davidson. Probabilistic predicate-aware modulo scheduling. In *International symposium on Code generation and optimization: feedback-directed and runtime optimization*, 2004.
25. Jian Wang, Christine Eisenbeis, Martin Jourdan, and Bogong Su. Decomposed software pipelining: a new perspective and a new approach. *Int. J. Parallel Program.*, 22(3):351–373, 1994.

Single machine scheduling with delivery dates and cumulative payoffs

Yasmina Seddik · Christophe Gonzales · Safia Kedad-Sidhoum

Abstract We address a scheduling problem with a new criterion and unequal release dates. This new criterion results from a practical situation in the domain of book digitization. Given a set of job-independent delivery dates, the goal is to maximize the cumulative number of jobs processed before each delivery date. We establish the complexity of the general problem. In addition, we provide a pseudopolynomial time algorithm for the two delivery dates case based on dynamic programming, and we discuss some polynomial cases.

1 Introduction

In this paper, we address a one machine non-preemptive scheduling problem where N jobs $J_1, \dots, J_N$ have to be scheduled. Each job J_i has a release date $r_i \geq 0$ and a processing time $p_i > 0$. K delivery dates are given: $D_1, \dots, D_K$, with $0 < D_1 < \dots < D_K$.

Given a schedule S , which is characterized by the completion times of all the jobs, variable V_k represents the number of jobs executed 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. [2], the problem we address in this paper can be defined as $1|r_i|\sum_{k=1}^K V_k$.

We will 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 before which J_i completes in S . Let C_i be the completion time of job J_i in S ; $v(J_i)$ can be represented by the following non-increasing stepwise function:

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

$$v(J_i) = \begin{cases} K & \text{if } 0 < C_i \leq D_1 \\ \vdots & \\ 2 & \text{if } D_{K-2} < C_i \leq D_{K-1} \\ 1 & \text{if } D_{K-1} < C_i \leq D_K \\ 0 & \text{if } D_K < C_i \end{cases}$$

In the example of Figure 1, jobs J_1 and J_2 are executed before D_1 , the payoff of each of these jobs is thus 3. J_4 's completion time occurs after D_1 but before D_2 , so J_4 's payoff is 2. Finally, J_3 's completion time occurs after D_2 but not later than D_3 : its payoff is 1. Consequently, the total payoff is 9.



Fig. 1 Example of a schedule

If we do not take into account the cumulative aspect of the payoffs, our problem is related to the problems with fixed delivery dates studied by Hall et al. [1]. They considered several classical scheduling criteria, including the following variant: the cost of a job J_i depends on the earliest delivery date occurring after the completion of J_i . In addition, they established complexity results for several problems, with different criteria and machine configurations.

As for the cumulative payoffs, Janiak and Krysiak [4] define a criterion which is a generalization of $\sum_{k=1}^K V_k$. They attempt to minimize a schedule's cost defined as the sum of the costs of the jobs in the schedule, each job having its own distinct stepwise cost function. Janiak and Krysiak show that the problem without release dates is NP-hard in the general case. They also provide a pseudopolynomial time algorithm for the special case in which the breakpoints are the same for all the jobs' stepwise functions.

The same criterion as in Janiak and Krysiak [4] is considered by Detienne et al. [5] for the unrelated parallel machines problem with release dates. They address a problem which is clearly a generalization of $1|r_i|\sum_{k=1}^K V_k$, and propose generic exact methods (Mixed Integer Programming, dedicated constraint generation procedure). Our contribution is to propose dedicated exact methods to solve $1|r_i|\sum_{k=1}^K V_k$.

Finally, there exists another class of problems closely related to $1|r_i|\sum_{k=1}^K V_k$: the generalized due date problem [3], where due dates are not related to the jobs. Instead, global due dates are defined, and before each of them, one job must complete. Complexity results have been established.

Problem $1|r_i|\sum_{k=1}^K V_k$ has a practical application in the domain of 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 and the manufacturer. We focus on the manufacturer's point of view, which tries to meet the client requirements while maximizing her own profits. The books to be digitized become available at different dates. In addition, the client sets several delivery dates D_k , $k = 1, \dots, K$ (such that $0 < D_1 < \dots < D_K$), and target quantities Q_k , $k = 1, \dots, K$. For a given $k = 1, \dots, K$, Q_k is the number of books the client expects to be digitized from the very beginning until date D_k . Therefore, the quantity Q_k includes the quantity Q_{k-1} plus an additional amount of expected books, and we have: $0 < Q_1 < Q_2 < \dots < Q_K$. From the preceding follows the cumulative aspect of the problem.

Because V_k is the number of books having been digitized from the very beginning until date D_k , it is desirable that V_k reaches Q_k if possible, or else, that V_k be as close as possible to Q_k . However, digitizing more than Q_k books before D_k increases the manufacturer's payoff (which is proportional to the number of digitized books). These goals can be achieved by maximizing the difference $V_k - Q_k$ for every $k = 1, \dots, K$. Aggregating these K criteria by summation, we must maximize the objective function $\sum_{k=1}^K (V_k - Q_k)$ or, equivalently, $\max \sum_{k=1}^K V_k$ as $\sum_{k=1}^K Q_k$ is a constant. This objective function also reflects the following preference of the manufacturer: earning a given payoff is more valuable earlier than later.

The paper is organized as follows. The NP-hardness of the general problem is discussed in Section 2. Section 3 describes four polynomial cases. Section 4 addresses the two delivery dates case. Finally, we draw some conclusions in Section 5.

2 Complexity of the general problem

In order to prove the NP-hardness of Problem 1 $|r_i| \sum_{k=1}^K V_k$, we rephrase it as the following decision problem:

MDD (Multiple Delivery Dates problem). Given a set of N jobs $J_1, \dots, J_N$, K delivery dates $D_1, \dots, D_K$, and a value V , does there exist a schedule S such that $v(S) \geq V$?

To prove the NP-completeness of MDD we will reduce to it a 3-partition problem, defined as follows [6]:

3-PARTITION. Given positive integers m, B , and a set of integers $A = \{a_1, a_2, \dots, a_{3m}\}$ such that $\sum_{i=1}^{3m} a_i = mB$ and $B/4 < a_i < B/2$ for $1 \leq i \leq 3m$, does there exist a partition $\langle A_1, A_2, \dots, A_m \rangle$ of A into 3-element sets such that, for each i , $\sum_{a \in A_i} a = B$?

Before proving the NP-completeness of MDD, we need to introduce the following definition.

Definition. In a schedule S , a straddling job J_i is a job such that $C_i - p_i < D_k < C_i$, for some $k \in \{1, \dots, K\}$.

Theorem 1 *MDD is unary NP-complete.*

Proof Given a feasible solution S for MDD, its payoff $v(S)$ can be computed in $O(N)$ time, thus $MDD \in NP$.

We show that 3-partition reduces to MDD. Suppose we are given an instance of 3-partition, with m, B and A . The corresponding input to MDD is a set of $N = 3m(m+2)$ jobs, a set of m delivery dates, and a value $V = (6+3m)m(m+1)/2$.

We define three different kinds of jobs, with processing times and release dates specified as follows. For each $i \in \{1, \dots, 3m\}$:

- job $\tilde{J}_i$ has a processing time of $mB + a_i$, and its release date is zero,
- job $\hat{J}_i$ has a processing time of mB , and its release date is zero,
- for $j \in \{1, \dots, m\}$, job $\bar{J}_{i,j}$ has a processing time of $\frac{1}{3m}$, and its release date is $jB(6m+1) + j - 1$.

The delivery dates are: $D_j = jB(6m+1) + j$, $j \in \{1, \dots, m\}$. We also set: $D_0 = 0$.

We show that MDD has a solution with a payoff at least equal to V if and only if the desired partition of A exists.

First, if there exists a partition $\langle A_1, A_2, \dots, A_m \rangle$ of A , such that for each i , $\sum_{a \in A_i} a = B$, then the following schedule σ is such that $v(\sigma) = V$ (see Figure 2). For each $j \in \{1, \dots, m\}$:

1. the three jobs $\{\tilde{J}_{1,j}, \tilde{J}_{2,j}, \tilde{J}_{3,j}\}$ (of length $mB + a_{1,j}, mB + a_{2,j}, mB + a_{3,j}$ respectively, where $A_j = \{a_{1,j}, a_{2,j}, a_{3,j}\}$) are scheduled from D_{j-1} to $D_{j-1} + B(3m + 1)$,
2. the three jobs $\hat{J}_{3(j-1)+1}, \hat{J}_{3(j-1)+2}, \hat{J}_{3(j-1)+3}$ are scheduled from $D_{j-1} + B(3m + 1)$ to $D_{j-1} + B(6m + 1) = D_j - 1$,
3. finally, the $3m$ jobs $\bar{J}_{1,j}, \dots, \bar{J}_{3m,j}$ are scheduled from $D_j - 1$ to D_j .

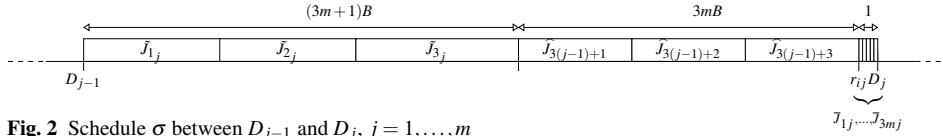


Fig. 2 Schedule σ between D_{j-1} and D_j , $j = 1, \dots, m$

In schedule σ , $6 + 3m$ jobs are executed between each pair of consecutive delivery dates. Hence, $v(\sigma) = (6 + 3m)(1 + 2 + \dots + m) = (6 + 3m)m(m + 1)/2 = V$.

Conversely, assume there exists a schedule σ' such that $v(\sigma') \geq V$.

There are two cases to be considered.

First, suppose that there is no straddling job in σ' . There can be at most $6 + 3m$ jobs scheduled before D_1 , in the interval $I_1 = [0, D_1]$. Actually, all the jobs with release dates equal to zero (all the $\tilde{J}$ -type and $\hat{J}$ -type jobs) have processing times at least equal to mB . So, at most 6 of them can be scheduled before D_1 . Moreover, at most $3m$ $\bar{J}$ -type jobs can be scheduled before D_1 . Because we assume there is no straddling job, the same reasoning remains valid for every time interval $I_j = [D_{j-1}, D_j]$, $j = 1, \dots, m$. Thus, σ' has at most $6 + 3m$ jobs scheduled in each interval I_j , $j = 1, \dots, m$. We deduce that σ' has to have exactly $6 + 3m$ jobs scheduled in each interval I_j , $j = 1, \dots, m$, otherwise $v(\sigma') < V$. We have shown that if σ' has no straddling job, it has exactly $6 + 3m$ jobs delivered at each delivery date, and $v(\sigma') = V$.

Second, suppose that σ' contains straddling jobs. Let D_{j_s} be the first delivery date with a straddling job J_s . Then the time interval between $D_{j_s} - 1$ and D_{j_s} is necessarily filled by the execution of J_s . Therefore, the jobs $\bar{J}_{1,j_s}, \dots, \bar{J}_{3m,j_s}$ cannot be scheduled before D_{j_s} . In the best case, they are executed before D_{j_s+1} . The action of postponing jobs $\bar{J}_{1,j_s}, \dots, \bar{J}_{3m,j_s}$ after D_{j_s} decreases the objective function value of an amount of at least $3m$. On the other hand, the straddling job increases the payoff of an amount of at most $m - 1$. Hence, the payoff of a solution with a straddling job cannot be greater than or equal to V .

In schedule σ' , for every $j \in \{1, \dots, m\}$, all the jobs $\bar{J}_{1,j}, \dots, \bar{J}_{3m,j}$ are scheduled between $D_j - 1$ and D_j . Otherwise, it would not be possible to schedule $6 + 3m$ jobs between D_{j-1} and D_j . The $6m$ remaining jobs (all the $\tilde{J}$ -type and $\hat{J}$ -type jobs) are distributed by groups of 6 jobs in each interval $I'_j = [D_{j-1}, D_j - 1]$, $j = 1, \dots, m$. The length of each interval I'_j is $B(6m + 1)$. All the $\tilde{J}$ -type and $\hat{J}$ -type jobs have lengths equal or greater than mB . Thus, each such job can be seen as composed of a first part whose length is mB , and a second part whose length is 0 for $\hat{J}$ -type jobs and a_i for $\tilde{J}$ -type jobs. If 6 jobs are scheduled in I'_j , then $6mB$ slots of time are filled by the first parts of these 6 jobs. The free slots of I'_j , whose total length is B , are occupied by the second parts of $\tilde{J}$ -type jobs, each one being of length a_i . The same reasoning holds for every I'_j , $j = 1, \dots, m$. Therefore, the second parts of $\tilde{J}$ -type jobs, each one of length a_i , are partitioned in m groups, each one having a total length of B , which constitutes a solution to 3-Partition. $\square$

3 Polynomial cases

Four polynomial cases have been identified, which result from relaxations of the general problem $1|r_i|\sum_{k=1}^K V_k$.

3.1 Relaxing the release dates

The problem with no release date $1||\sum_{k=1}^K V_k$ can be easily shown to be optimally solved by sequencing the jobs in a nondecreasing order of their processing times (SPT rule).

3.2 Preemptive case

In order to show how to solve the problem $1|r_i, pmtn|\sum_{k=1}^K V_k$, we use the Shortest Remaining Processing Time rule (SRPT), which is also called Modified Smith's rule [7]: at each release time or finishing time of a job, schedule an unfinished job which is available and has the smallest remaining processing time.

An optimal solution for problem $1|r_i, pmtn|\sum V_k$ can be obtained using the SRPT rule. This can be proved with the same arguments used to show that applying SRPT [7] yields an optimal solution for the problem $1|r_j, pmtn|\sum C_j$.

Therefore, the problem $1|r_i, pmtn|\sum_{k=1}^K V_k$ can be solved in $O(N \log N)$ time.

3.3 Identical processing times

We consider the case where all the jobs have the same processing time $1|r_i, p_i = p|\sum_{k=1}^K V_k$.

Definition. A “left-shifted” solution is a solution such that each job starts as soon as possible. This means that each job starts at its release date if it is possible, otherwise at the completion time of the previous job in the schedule. Hence, to represent a left-shifted solution S it is sufficient to provide the ordered sequence of all the jobs in S (including those that are scheduled after D_K).

Proposition 1 *A left-shifted solution in which jobs are ordered following a nondecreasing order of their release dates is optimal for the problem $1|r_i, p_i = p|\sum_{k=1}^K V_k$.*

Proof In the following proof, we will only consider left-shifted solutions. Indeed, every feasible solution can be transformed into a left-shifted solution in $O(N)$ time, without decreasing its payoff.

Let S_R be a solution in which jobs are ordered following a nondecreasing order of their release dates. We show that every feasible solution S can be transformed into S_R , and that $v(S_R) \geq v(S)$. We renumber the jobs following their order in S : $J_1, \dots, J_N$. If there exists a job J_i such that $r_i > r_{i+1}$, or that $r_i = r_{i+1}$ and J_i is scheduled after J_{i+1} in S_R , then by swapping J_i and J_{i+1} in S , the total payoff will not decrease. Indeed, swapping two jobs that have the same release date does not change the total payoff, since $p_i = p_{i+1}$. Moreover, if $r_i > r_{i+1}$, swapping J_i and J_{i+1} can strictly improve the total payoff, if in S , J_i starts at r_i and there is an idle time in the time interval $[r_i - 1, r_i]$. Using this process, we get, after a polynomial number of steps, the solution S_R such that $v(S_R) \geq v(S)$. Because this result is true for every feasible solution S , it is also true for every optimal solution S^* where $v(S_R) \geq v(S^*)$. Hence the proposition is true. $\square$

Therefore, the problem $1|r_i, p_i = p|\sum_{k=1}^K V_k$ can be solved in $O(N \log N)$ time.

3.4 Single delivery date

The problem with a single delivery date D , $1|r_i|V$, is equivalent to the problem $1|r_i, d_i = d|\sum U_i$. By definition, every job J_i can be processed in the time period $F_i = [r_i, d]$. We call an “on time” job, every job J_i that is processed in the time interval F_i . The jobs that are not on time can be scheduled after d , in any order.

Lemma 1 *There exists an optimal solution for $1|r_i, d_i = d|\sum U_i$ such that the last on time job ends at time d .*

Proof Let S^* be an optimal solution such that the last on time job does not end at time d . Let J_l be the last on time job and let C_l be its completion time in S^* . We can move J_l to the right, in order to make it finish at time d . The jobs that follow J_l in S^* are also moved toward right of $d - C_l$ time units, in order to avoid jobs overlapping; the payoff of the new schedule is identical to that of S^* . □

Lemma 2 *There exists an optimal solution for $1|r_i, d_i = d|\sum U_i$ such that there is no idle time between any pair of consecutive on time jobs.*

The arguments of the proof are similar to the ones of Lemma 1 (jobs are moved to the right).

We provide a polynomial time algorithm to solve $1|r_i, d_i = d|\sum U_i$, which is based on the Moore-Hodgson algorithm [9] for solving $1||\sum U_i$.

We renumber jobs from 1 to N , following the nonincreasing order of their release dates.

S is an ordered set of jobs, which, at the end of the algorithm, will represent an optimal schedule. At each iteration, the jobs of S satisfy the condition described in Lemma 1.

We will use the following notations:

- $\forall J_i \notin S$, $J_i.S$ denotes the sequence obtained by adding J_i at the beginning of S .
- $\forall J_j \in S$, $S \setminus J_j$ denotes the sequence obtained by removing J_j from S , leaving all the other jobs in the order given by S .

Input: $J_1, \dots, J_N$, ordered following their nonincreasing release dates

Output: S

```

1  $S \leftarrow \emptyset, t \leftarrow d$ 
2 for  $i = 1$  to  $N$  do
3    $S \leftarrow J_i.S$ 
4    $t \leftarrow t - p_i$ 
5   if  $t < r_i$  then
6      $J_j \leftarrow$  job in  $S$  with the largest processing time
7      $S \leftarrow S \setminus J_j$ 
8      $t \leftarrow t + p_j$ 
9 complete  $S$ : add at its end all the jobs  $J_i \notin S$  (for  $i = 1, \dots, N$ ), in any order
10 return  $S$ 
```

Algorithm 1: Algorithm for $1|r_i, d_i = d|\sum U_i$

Theorem 2 *Algorithm 1 solves optimally the problem $1|r_i, d_i = d|\sum U_i$.*

Proof The arguments for the proof of correctness are similar to those for the Moore-Hodgson algorithm [8]. We give a sketch of proof below.

At each iteration, the on time jobs satisfy the condition described in Lemma 1. The jobs are numbered from 1 to N following the order of their nonincreasing release dates. We call S_{1m} the sequence $J_1, \dots, J_m$. A subsequence E of S_{1m} is feasible, if all the jobs of E can be processed on time, in the order given by E . Given a sequence S , its processing time $p(S)$ is defined as: $p(S) = \sum_{J_i \in S} p_i$. For each $m \in \{1, \dots, N\}$:

- Let E_m be a feasible subsequence of S_{1m} with the maximum number of on time jobs. We set $N_m = |E_m|$.
- Let S_m be a feasible subsequence of S_{1m} , such that $|S_m| = N_m$, and $p(S_m)$ is minimal.

The sequence S_N obtained at the end of the algorithm is optimal. The proof is made by induction.

It is trivial to see that the algorithm correctly constructs S_1 . The induction hypothesis is that the algorithm indeed constructs S_m . We want to prove that sequence S_{m+1} can be constructed starting out with sequence S_m .

At the $(m+1)$ -th iteration of the algorithm, job J_{m+1} is added at the beginning of S_m . There are two cases to be considered.

Case 1. J_{m+1} is processed into F_{m+1} .

We have $N_{m+1} = N_m + 1$, because $|J_{m+1}.S_m| = N_m + 1$ and $|J_{m+1}.S_m|$ cannot be greater than $N_m + 1$ (by induction hypothesis on $|S_m| = N_m$).

Moreover, every feasible subsequence of S_{1m+1} that contains N_{m+1} jobs, includes job J_{m+1} . Indeed, by induction hypothesis, every feasible subsequence S' of S_{1m} satisfies $|S'| \leq N_m$. Then, to construct S_{m+1} , we must add job J_{m+1} to a feasible subsequence S'' of S_{1m} with N_m jobs and minimal processing time. $S'' = S_m$ by induction hypothesis, hence $p(J_{m+1}.S_m)$ is minimal among the processing times of the feasible sequences of N_{m+1} jobs.

Case 2. J_{m+1} is not processed into F_{m+1} .

We have $N_{m+1} = N_m$. Indeed, J_{m+1} cannot belong to a feasible subsequence S''' of S_{1m+1} such that $|S'''| = N_m + 1$, because by adding J_{m+1} to S_m (which contains N_m jobs and has the shortest processing time), J_{m+1} cannot be on time. Thus, N_m is the maximal number of jobs of a feasible subsequence of S_{1m+1} .

Moreover, adding J_{m+1} to S_m does not increase the number of on time jobs. However, adding J_{m+1} to S_m and deleting the job with longest processing time among $S_m \cup \{J_{m+1}\}$ keeps the same number N_m of on time jobs, and reduces as much as possible the total processing time of the sequence. If J_{m+1} has the largest processing time among $S_m \cup \{J_{m+1}\}$, then $S_{m+1} = S_m$. Hence, the algorithm constructs a feasible subsequence with minimal processing time. □

Algorithm 1 runs in $O(N \log N)$ if a heap is used for the search of the largest job.

4 The problem with two delivery dates

We consider the problem of $1|r_i|\sum_{k=1}^K V_k$ with $K = 2$.

4.1 Complexity of the problem

In order to prove the NP-hardness of $1|r_i|V_1 + V_2$, we have to prove the NP-completeness of the corresponding decision problem, defined as follows.

2DD (Two Delivery Dates problem). Given a collection of N jobs: $\{J_1, \dots, J_N\}$, two delivery dates: D_1, D_2 , and a value V , does there exist a schedule S such that $v(S) \geq V$?

We show a polynomial reduction from Partition problem to 2DD. A definition of the Partition problem follows.

Partition. Given n positive integers $s_1, \dots, s_n$, does there exist $L' \subset L = \{1, \dots, n\}$ such that: $\sum_{i \in L'} s_i = \sum_{i \in L \setminus L'} s_i$?

Theorem 3 2DD is NP-complete.

Proof Given a feasible solution S for 2DD, its payoff $v(S)$ can be computed in $O(N)$ time, thus $2DD \in NP$.

We show that Partition reduces to 2DD. Suppose we are given an instance of Partition, with n positive integers $s_1, \dots, s_n$. Let $b = \frac{1}{2} \sum_{i=1}^n s_i$. The corresponding input of 2DD is a set of $3n$ jobs, two delivery dates and a value $V = 5n$.

There are three different kinds of jobs, with processing times and release dates specified as follows. For each $i \in \{1, \dots, n\}$:

- job $\tilde{J}_i$ has a processing time of $b + s_i$, and its release date is zero,
- job $\hat{J}_i$ has a processing time of b , and its release date is zero,
- job $\bar{J}_i$ has a processing time of $\frac{1}{n}$, and its release date is $b(n+1)$.

The delivery dates are: $D_1 = b(n+1) + 1$ and $D_2 = 2b(n+1) + 1$.

We now have to show that 2DD has a solution of value at least V if and only if the desired partition of $s_1, \dots, s_n$ exists.

First, if Partition has a solution, the solution is represented by a partition of L in two sets L' and $L \setminus L'$. Let k be the cardinality of L' . The cardinality of $L \setminus L'$ is then $n - k$.

Let σ be the following schedule.

1. The k jobs of the set $\{\tilde{J}_i | i \in L'\}$ are executed from time zero to $b(k+1)$
2. The $n - k$ jobs of the set $\{\hat{J}_i | i \in L \setminus L'\}$ are executed from $b(k+1)$ to $b(n+1)$
3. The n jobs $\bar{J}_1, \dots, \bar{J}_n$ are executed from $b(n+1)$ to $b(n+1) + 1$
4. The $n - k$ jobs $\{\tilde{J}_i | i \in L \setminus L'\}$ are executed from $b(n+1) + 1$ to $b(n+1) + 1 + b(n - k + 1) = 1 + b(2n - k + 2)$
5. The k jobs of the set $\{\hat{J}_i | i \in L'\}$ are executed from $1 + b(2n - k + 2)$ to $2b(n+1) + 1$

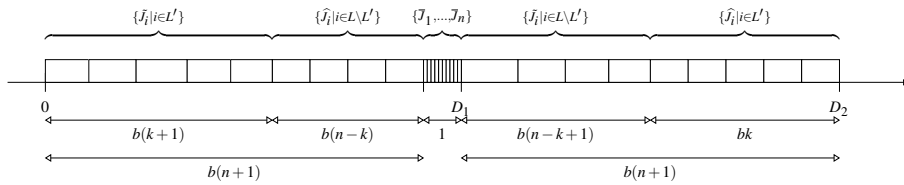


Fig. 3 Schedule σ

Therefore, in schedule σ , $k + (n - k) + n = 2n$ jobs are executed before D_1 , and $(n - k) + k = n$ jobs have their completion time before D_2 and after D_1 . Hence, σ 's value is $5n$.

Conversely, suppose there exists a schedule σ' whose value is $5n$. We first prove that, in σ' , $2n$ jobs are executed before D_1 . Indeed, if less than $2n$ jobs were executed before D_1 , then σ' 's value would be less than $5n$ (because of the total number of jobs). Hence, the n jobs $\bar{J}_1, \dots, \bar{J}_n$ must all be executed before D_1 , otherwise it would be impossible to execute $2n$ jobs before D_1 , since all the other jobs have processing times larger than b . Thus, at most n other jobs can be executed before D_1 . Indeed, all the $\tilde{J}$ -type and $\hat{J}$ -type have processing times at least equal to b . Therefore, at most n of them can be executed in the interval $[0, b(n+1)]$ (the total number of $\hat{J}$ -type jobs is n).

So, in schedule σ' , we have exactly n jobs executed in the interval $I_1 = [0, b(n+1)]$ and n jobs executed in the interval $I_2 = [D_1, D_2]$. The length of each of these intervals is $b(n+1)$. As we said, all the $\tilde{J}$ -type and $\hat{J}$ -type jobs have processing times at least equal to b . Hence, each of them can be seen as composed of two parts: a first part of length b , and a second part whose length is 0 for $\hat{J}$ -type jobs and s_i for $\tilde{J}$ -type jobs. Because n jobs are executed in I_1 , nb slots of time are occupied by the first parts of these n jobs. The remaining portion of I_1 , which is of length b , is occupied by the second parts of $\tilde{J}$ -type jobs, each of length s_i . The same reasoning holds for I_2 . Therefore, the second parts of the $\tilde{J}$ -type jobs, each of length s_i , are partitioned in two groups, each of total length equal to b , which constitutes a solution to Partition. $\square$

In section 4.2 we present a pseudopolynomial time algorithm for $1|r_i|V_1 + V_2$, proving thus that $1|r_i|V_1 + V_2$ is weakly NP-hard.

4.2 Dynamic programming algorithm

Any schedule S of $1|r_i|V_1 + V_2$ can be split into three subschedules: the subschedule S_1 of the jobs whose completion times are in the interval $I_1 = [0, D_1]$, the subschedule S_2 of the jobs whose completion times are in the interval $I_2 =]D_1, D_2]$ and the subschedule S_3 of the jobs whose completion times are in the interval $I_3 =]D_2, +\infty[$. We have $S = S_1.S_2.S_3$, where $S_i.S_j$ denotes the concatenation of subschedule S_i with subschedule S_j .

We first introduce two lemmas that will be useful for the proof of correctness of the algorithm.

Property 1 Given a feasible solution $S = S_1.S_2.S_3$, all the jobs of S_1 (respectively S_2) are ordered in non-decreasing order of their release dates.

Lemma 3 *There exists an optimal solution of $1|r_i|V_1 + V_2$ that satisfies Property 1.*

Proof We first show that every feasible solution $S = S_1.S_2.S_3$ can be modified in order to obtain a solution S_R that satisfies Property 1 and such that $v(S_R) \geq v(S)$. Let J_i and J_j be two consecutive jobs in S , such that:

- J_i and J_j belong both to S_1 or both to S_2 ,
- $r_i > r_j$ and
- $C_i < C_j$.

Let $S' = S'_1.S'_2.S'_3$ be a solution obtained from S by swapping J_i and J_j without moving the other jobs, in such a way that $C'_i = C_j$ and $C'_j = C_i - p_i + p_j$. Two cases need to be considered:

Case 1. If J_i and J_j belonged to S_1 , then $C'_j \in I_1$ and $C'_i \in I_1$. Thus, J_i and J_j belong to S'_1 and $v(S') = v(S)$.

Case 2. If J_i and J_j belonged to S_2 , two subcases may occur:

- $C'_j > D_1$: then, $C'_j \in I_2$ and $C'_i \in I_2$. Thus, J_i and J_j belong to S'_2 and $v(S') = v(S)$.
- $C'_j \leq D_1$: then, $C'_j \in I_1$ and $C'_i \in I_2$. Thus, $J_i \in S'_2$ and $J_j \in S'_1$ and $v(S') = v(S) + 1$.

Iterating this process, we get after a finite number of steps (bubble sort algorithm) a feasible solution S_R that satisfies Property 1.

In particular, this modification process can be applied to any optimal solution. Hence the result. $\square$

Property 2 Given a solution $S = S_1.S_2.S_3$, there is no idle time between any pair of consecutive jobs in S_2 .

Lemma 4 *There exists an optimal solution that satisfies Property 2.*

Proof Let $S^* = S_1^*.S_2^*.S_3^*$ be an optimal solution that does not satisfy Property 2, and let k_2 be the completion time of the last job of S_2^* . Without moving the last job of S_2^* , we can move the other jobs of S_2^* as far as possible to the right, starting from the last but one until the first one. Then, each of the jobs of S_2^* (except the last one) ends exactly at the start time of the next job. We obtain a solution of value $v(S^*)$, because the jobs of S_1^* and S_3^* do not move, and the jobs of S_2^* still complete in the interval I_2 . Hence, the lemma holds. $\square$

Corollary 1 *There exists an optimal solution that satisfies Properties 1, 2.*

The proof is obvious because the arguments of the proofs of Lemmas 3 and 4 can be combined.

Algorithm 2, which is based on dynamic programming, constructs schedules that satisfy Properties 1, 2, by adding iteratively jobs before D_1 , between D_1 and D_2 , or after D_2 . For any schedule $S = S_1.S_2.S_3$, we define the following notations:

- for any job J_j in S , the completion time of J_j in S is denoted by $C_j(s)$;
- the completion time of the last job of S_1 is denoted by $k_1(S)$ or, when no ambiguity is possible, by k_1 ;
- the starting time of the first job of S_2 is denoted by $d(S)$ or, when no ambiguity is possible, by d ;
- the completion time of the last job of S_2 is denoted by $k_2(S)$ or, when no ambiguity is possible, by k_2 .

Figure 4 illustrates these notations on an example. Schedules can be summarized by their states e defined as $e = \langle k_1, d, k_2 \rangle$.

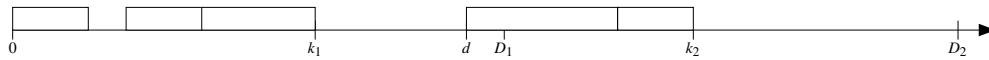


Fig. 4 Definition of k_1 , k_2 and d

In Algorithm 2, we handle pairs (e, u) where e is a state, and u is the payoff of the best of all the constructed schedules that correspond to the state e . In addition, for each job J_j , we define $b_j = \max(r_j, D_1 - p_j + 1)$, i.e., b_j is the earliest starting time allowing job J_j to complete after D_1 .

Input: p_i, r_i (jobs sorted by nondecreasing r_i), D_1, D_2
Output: The payoff of an optimal schedule

```

1  $\mathcal{S}_0 \leftarrow \{(\langle 0, +\infty, 0 \rangle, 0)\}$ 
2 for  $j = 1$  to  $N$  do
    // Adding task  $J_j$  to  $S_3$ 
3    $\mathcal{S}_j \leftarrow \mathcal{S}_{j-1}$ 
    // Adding task  $J_j$  to  $S_1$ 
4   foreach  $(\langle k_1, d, k_2 \rangle, v) \in \mathcal{S}_{j-1}$  do
5      $\alpha \leftarrow \max(r_j, k_1) + p_j$ 
6      $S \leftarrow \begin{cases} \langle \alpha, d, k_2 \rangle & \text{if } \alpha \leq d \\ \langle \alpha, \alpha, k_2 + \alpha - d \rangle & \text{otherwise} \end{cases}$ 
7     if  $k_1(S) \leq D_1$  and  $k_2(S) \leq D_2$  then
8        $\mathcal{S}_j \leftarrow \begin{cases} \mathcal{S}_j \cup \{(S, v+2)\} \setminus \{(S, v')\} & \text{if } \exists (S, v') \in \mathcal{S}_j \text{ such that } v' < v+2 \\ \mathcal{S}_j & \text{if } \exists (S, v') \in \mathcal{S}_j \text{ such that } v' \geq v+2 \\ \mathcal{S}_j \cup \{(S, v+2)\} & \text{otherwise} \end{cases}$ 
    // Adding task  $j$  to  $S_2$ 
9   foreach  $(\langle k_1, d, k_2 \rangle, v) \in \mathcal{S}_{j-1}$  do
10     $\alpha \leftarrow \max(r_j, b_j, k_1, k_2) + p_j$ 
11     $S \leftarrow \begin{cases} \langle k_1, \alpha - p_j, \alpha \rangle & \text{if } d = +\infty \\ \langle k_1, d + \alpha - p_j - k_2, \alpha \rangle & \text{otherwise} \end{cases}$ 
12    if  $k_1(S) \leq D_1$  and  $k_2(S) \leq D_2$  then
13       $\mathcal{S}_j \leftarrow \begin{cases} \mathcal{S}_j \cup \{(S, v+1)\} \setminus \{(S, v')\} & \text{if } \exists (S, v') \in \mathcal{S}_j \text{ such that } v' < v+1 \\ \mathcal{S}_j & \text{if } \exists (S, v') \in \mathcal{S}_j \text{ such that } v' \geq v+1 \\ \mathcal{S}_j \cup \{(S, v+1)\} & \text{otherwise} \end{cases}$ 
14 return  $\max\{v : (S, v) \in \mathcal{S}_N\}$ 

```

Algorithm 2: Pseudopolynomial algorithm

It is easy to compute an optimal solution by adding some variables in the state definition, without deteriorating the time complexity of the algorithm. However, for the sake of clarity, we present here a simpler version which only returns the optimal payoff.

Theorem 4 *Algorithm 2 returns the payoff of an optimal schedule.*

Proof Let $\mathcal{S}$ denote the set of all the possible schedules satisfying Properties 1, 2. Let $\mathcal{E}$ denote the set of all states and let $f : \mathcal{S} \mapsto \mathcal{E} \times \mathbb{R}$ be the function assigning to any schedule S a pair (e, u) where e and u represent S 's state and payoff respectively. Finally, let $\mathcal{Q}_0 = \{(\langle 0, +\infty, 0 \rangle, 0)\}$ and, for any $j \in \{1, \dots, N\}$, let $\mathcal{Q}_j = \cup_{(e, u) \in \mathcal{Q}_{j-1}} (g_0(e, u, j) \cup g_1(e, u, j) \cup g_2(e, u, j))$ where g_0, g_1, g_2 are three functions defined from $\mathcal{E} \times \mathbb{R} \times \{1, \dots, N\}$ to $2^{\mathcal{E} \times \mathbb{R}}$. g_0 (resp. g_1 , resp. g_2) associates to each pair (e, u) (which corresponds to at least one schedule $S = S_1.S_2.S_3$) of $\mathcal{Q}_{j-1}$ (for $j = 0, \dots, N-1$) the state (e', u') of $\mathcal{Q}_j$, corresponding to the schedule obtained from S by adding job j to S_3 (resp. S_1 , resp. S_2). g_0, g_1, g_2 are defined as follows :

$$\begin{aligned}
 g_0(e, u, j) &= \{(e, u)\} \\
 g_1(\langle k_1, d, k_2 \rangle, u, j) &= \\
 &\begin{cases} \{(\langle \alpha, d, k_2 \rangle, u+2)\} & \text{if } \alpha = \max(r_j, k_1) + p_j \leq \min(d, D_1) \\ \{(\langle \alpha, \alpha, k_2 + \alpha - d \rangle, u+2)\} & \text{if } d < \alpha = \max(r_j, k_1) + p_j \leq D_1 \text{ and } k_2 + \alpha - d \leq D_2 \\ \emptyset & \text{otherwise} \end{cases} \\
 g_2(\langle k_1, d, k_2 \rangle, u, j) &= \\
 &\begin{cases} \{(\langle k_1, \alpha - p_j, \alpha \rangle, u+1)\} & \text{if } d = +\infty \text{ and } \alpha = \max(r_j, b_j, k_1) + p_j \leq D_2 \\ \{(\langle k_1, d + \alpha - p_j - k_2, \alpha \rangle, u+1)\} & \text{if } \alpha = \max(b_j, r_j, k_2) \leq D_2 \\ \emptyset & \text{otherwise} \end{cases}
 \end{aligned}$$

We will show by induction that there exist some pairs $(e, u) \in \mathcal{Q}_N$ corresponding to some optimal schedules. Let S^* be an optimal schedule satisfying Properties 1 and 2. Let us denote by $\{J_{i_1}, \dots, J_{i_l}\}$, $i_1 < i_2 < \dots < i_l$, the set of jobs of S^* that complete before D_2 .

For $k = 1, \dots, l$, let S_{i_k} be the schedule that satisfies the following conditions, with minimum k_1 and k_2 (among all the schedules satisfying the same conditions) :

1. S_{i_k} schedules all the jobs $J_{i_1}, \dots, J_{i_k}$ before D_2 , and all the other jobs after D_2
2. if $C_{i_q}(S^*) \leq D_1$, then $C_{i_q}(S_{i_k}) \leq D_1$, $\forall q \in \{1, \dots, k\}$
3. if $D_1 < C_{i_q}(S^*) \leq D_2$, then $D_1 < C_{i_q}(S_{i_k}) \leq D_2$, $\forall q \in \{1, \dots, k\}$
4. S_{i_k} satisfies Properties 1 and 2

We show that, for every $k \in \{1, \dots, l\}$, $f(S_{i_k}) \in \mathcal{Q}_{i_k}$. As a consequence, we show that $f(S_{i_l}) \in \mathcal{Q}_{i_l} \subseteq \mathcal{Q}_N$.

The only state of set $\mathcal{Q}_0$ corresponds to a schedule with no jobs before D_2 . If $i_1 > 1$, for all $i \in \{1, \dots, i_1 - 1\}$: $(\langle 0, +\infty, 0 \rangle, 0) \in \{g_0(e, u, i) : (e, u) \in \mathcal{Q}_{i-1}\} \subseteq \mathcal{Q}_i$, by definition of $\mathcal{Q}_i$.

Let us assume that $C_{i_1}(S) \leq D_1$. Then, $(\langle r_{i_1} + p_{i_1}, +\infty, 0 \rangle, 2) = g_1(\langle 0, +\infty, 0 \rangle, 0, i_1) \in \{g_1(e, u, i_1) : (e, u) \in \mathcal{Q}_{i_1-1}\} \subseteq \mathcal{Q}_{i_1}$. This means that we create the schedule S_{i_1} , containing job J_{i_1} executed at time $r_{i_1} + p_{i_1}$, which is the earliest possible completion time for J_{i_1} , and all the other jobs executed after D_2 ; and $f(S_{i_1}) \in \mathcal{Q}_{i_1}$.

Assume now that $D_1 < C_{i_1}(S) \leq D_2$. Then, $(\langle 0, \max(r_{i_1}, b_{i_1}), \max(r_{i_1}, b_{i_1}) + p_{i_1} \rangle, 1) = g_2(\langle 0, +\infty, 0 \rangle, 0, i_1) \in \{g_2(e, u, i_1) : (e, u) \in \mathcal{Q}_{i_1-1}\} \subseteq \mathcal{Q}_{i_1}$. Here again, job J_{i_1} is executed at its earliest possible completion time $\max(r_{i_1}, b_{i_1})$.

Overall, we constructed schedule S_{i_1} satisfying conditions 1, 2, 3, 4, with minimum k_1 and k_2 , and such that $f(S_{i_1}) \in \mathcal{Q}_{i_1}$.

Assume now that we created the sequence $\{S_{i_k} : k \in \{1, \dots, j-1\}\}$ of schedules satisfying conditions 1, 2, 3, 4, with minimum k_1 and k_2 , and such that $f(S_{i_{j-1}}) \in \mathcal{Q}_{i_{j-1}}$.

Let $f(S_{i_{j-1}}) = (\langle k_1, d, k_2 \rangle, u)$.

Now assume that $C_{i_j}(S^*) \leq D_1$. Then, since k_1 is minimal, job J_{i_j} must start after both k_1 and r_{i_j} in S^* . So, the earliest possible starting time of job J_{i_j} is $\max(r_{i_j}, k_1)$. It may however happen that $\max(r_{i_j}, k_1) + p_{i_j} > d$ or, in other words, that job J_{i_j} overlaps the first job whose completion time is between D_1 and D_2 in $S_{i_{j-1}}$. To avoid this, we must shift all the jobs of $S_{i_{j-1}}$ starting after d of $\max(r_{i_j}, k_1) + p_{i_j} - d$. Note that such a shift is possible since: if a job can be executed at time t , it can always be executed later (there are constraints only on release dates); and $\max(r_{i_j}, k_1) + p_{i_j} - d$ is the minimal shift enabling job J_{i_j} and all the other jobs of $S_{i_{j-1}}$ to be executed. So this shift cannot be larger than that needed to obtain S^* .

Overall we get the schedule S_{ij} satisfying conditions 1, 2, 3, 4, with minimum k_1 and k_2 , and such that $f(S_{ij}) \in \mathcal{Q}_{ij}$.

Finally, assume that $D_1 < C_{ij}(S^*) \leq D_2$. Then the earliest possible starting time of J_{ij} in S_{ij} is clearly $\max(r_{ij}, b_{ij}, k_1)$ if it is the first one that completes between D_1 and D_2 , otherwise it is $\max(r_{ij}, b_{ij}, k_2)$. Function g_2 places the job precisely at that date and, to satisfy Property 2, it shifts all the jobs that complete between D_1 and D_2 so that this subschedule contains no idle time. Again, this produces a schedule S_{ij} satisfying conditions 1, 2, 3, 4, with minimum k_1 and k_2 , and such that $f(S_{ij}) \in \mathcal{Q}_{ij}$.

By induction, we just constructed a sequence of schedules ensuring that $v(S_{ij}) = v(S^*)$ (optimal payoff) and $f(S_{ij}) \in \mathcal{Q}_{ij} \subseteq \mathcal{Q}_N$. In addition, each time a job was added before D_1 (resp. between D_1 and D_2), the payoffs were increased by 2 (resp. 1). Hence, for any pair $(e, u) \in \mathcal{Q}_N$, u actually corresponds to the payoff of the schedules corresponding to state e . Consequently, some pairs $(e, u) \in \mathcal{Q}_N$ correspond to some optimal schedules and u is their payoff.

To conclude this proof, observe that if some $\mathcal{Q}_j$ contains two pairs $(\langle d_1, k, d_2 \rangle, u)$ and $(\langle d_1, k, d_2 \rangle, u')$ with $u' \leq u$, then removing pair $(\langle d_1, k, d_2 \rangle, u')$ cannot prevent to have pairs corresponding to some optimal schedules in $\mathcal{Q}_N$. Actually, assume that there exists a pair $(e, u'') \in \mathcal{Q}_N$ obtained by applying recursively functions g_0, g_1, g_2 on $(\langle d_1, k, d_2 \rangle, u')$. Then, by definition of these functions, the same applications of g_0, g_1, g_2 can be used on $(\langle d_1, k, d_2 \rangle, u)$, hence resulting in a pair $(e, u''') \in \mathcal{Q}_N$. Clearly, the states obtained from these applications are the same and only their payoffs can differ. As the applications will increase u and u' by the same value, if $u' \leq u$ then $u'' \leq u'''$. So discarding $(\langle d_1, k, d_2 \rangle, u')$ from $\mathcal{Q}_j$ cannot prevent obtaining an optimal solution.

Let $h : 2^{\mathcal{E}} \times \mathbb{R} \mapsto 2^{\mathcal{E}} \times \mathbb{R}$ be defined as $h(\mathcal{R}) = \{(e, u) \in \mathcal{R} : \nexists (e, u') \in \mathcal{R} \text{ such that } u' \geq u\}$. Let $\mathcal{S}_0 = \{(\langle 0, +\infty, 0 \rangle, 0)\}$ and, for any $j \in \{1, \dots, N\}$, $\mathcal{S}_j = \cup_{(e, u) \in \mathcal{S}_{j-1}} (h(g_0(e, u, j)) \cup h(g_1(e, u, j)) \cup h(g_2(e, u, j)))$. Then, by the preceding paragraph, there exist some pairs $(e, u) \in \mathcal{S}_N$ corresponding to some optimal schedules. Algorithm 2 computes precisely sets $\mathcal{S}_j$, $j = \{1, \dots, N\}$: Line 3 clearly computes $\{h(g_0(e, u, j)) : (e, u) \in \mathcal{S}_{j-1}\}$, Lines 5-8 compute $\{h(g_1(e, u, j)) : (e, u) \in \mathcal{S}_{j-1}\}$ and Lines 9-13 compute $\{h(g_2(e, u, j)) : (e, u) \in \mathcal{S}_{j-1}\}$. So the output of Algorithm 2 corresponds to the payoff of an optimal solution. $\square$

Proposition 2 *Algorithm 2 computes the optimal payoff of an instance in pseudopolynomial time ($O(N(D_1(D_2)^2) + N \log N)$).*

Proof First, observe that, by lines 8 and 13, it is impossible that some $\mathcal{S}_j$ contains two pairs (e, u) and (e', u') with $e = e'$. So the number of elements in each $\mathcal{S}_j$ is bounded by the number of states. Clearly, k_1 can only range from 0 to D_1 and k_2 from $b = \min_j \{b_j\}$ to D_2 . Similarly, the jobs ending between D_1 and D_2 start at time d , d can only range from b to D_2 . Overall, the number of states in each $\mathcal{S}_j$ is bounded by $X = (D_1 + 1) \times (D_2 - b + 1) \times (D_2 - b + 1)$. In each for loop of lines 2-13, we first copy $\mathcal{S}_{j-1}$ into $\mathcal{S}_j$, hence inducing a complexity of $O(X)$, then for each loop of lines 4-8, we try to add job J_j at the end of each schedule of $\mathcal{S}_{j-1}$, hence inducing an overall complexity of $O(X)$ to create states S on line 6 and, by storing $\mathcal{S}_j$ as an array or a hash table, a complexity of $O(X)$ is induced to update $\mathcal{S}_j$ on line 8. The same complexity clearly applies for the foreach loop of lines 9-13. So, overall, the complexity of executing lines 3-13 is $O(X) = O(D_1(D_2 - b)^2)$. The for loop of lines 2-13 is executed N times. Finally, sorting the jobs in nondecreasing release dates order can be achieved in $O(N \log N)$. Overall, the complexity of Algorithm 2 is $O(N(D_1(D_2 - b)^2) + N \log N)$ that is $O(N(D_1(D_2)^2) + N \log N)$. $\square$

5 Conclusions

This paper introduces a single machine scheduling problem with a new criterion based on cumulative payoffs regarding a set of delivery dates. Release dates are considered for the jobs. The strong NP-hardness of the general problem is established, a pseudopolynomial time algorithm is developed for the two delivery dates case, and some polynomial cases are discussed. They extend the complexity results known for one-machine scheduling problems for which the number of late jobs is minimized, release dates are considered and a common due date is imposed.

Further research will be devoted to find a branch-and-bound algorithm in order to solve the general problem. Some bounds and some dominance rules could be derived from the results we get for the polynomial cases, and especially from the algorithm of Section 3.3. Furthermore, a heuristic approach could be developed to obtain good solutions for the real practical instances of the problem.

References

1. N. G. Hall, M. Lesaoana and C. N. Potts, *Scheduling with Fixed Delivery Dates*, Operations Research (2001), 49, 134–144.
2. R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kann, *Optimization and approximation in deterministic sequencing and scheduling: A survey*, Annals of Discrete Mathematics (1979), 6, 287–326.
3. N. G. Hall, S. P. Sethi and C. Sriskandarajah, *On the complexity of generalized due date scheduling problems*, European Journal of Operational Research (1991), 51, 100–109.
4. A. Janiak and T. Krysiak, *Single processor scheduling with job values depending on their completion times*, Journal of Scheduling (2007), 10, 129–138.
5. B. Detienne, S. Dauzères-Pérès and C. Yugma *Scheduling jobs on parallel machine to minimize a regular step total cost function*, Journal of Scheduling (2011).
6. M. R. Garey, D. S. Johnson and R. Sethi, *The complexity of flowshop and jobshop scheduling*, Mathematics of Operations Research (1976), 1, 117–129.
7. P. Brucker, *Scheduling Algorithms*, Springer (2007), Berlin.
8. M. Pinedo, *Scheduling Theory, Algorithms, and Systems*, Prentice Hall (1995), Englewood Cliffs, New Jersey.
9. J. M. Moore, *An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs*, Management Science (1968), 15, 102–109.

Single server scheduling problem

Optimal policy for convex costs depends on arrival rates

Carlos F. Bispo

Abstract Being probably one of the oldest decision problems in queuing theory, the single server scheduling problem continues to be a challenging one. The original formulations considered linear costs and the resulting policy is puzzling in many ways. The main one is that, either for preemptive or non preemptive problems, it results in a priority ordering of the different classes of customers being served that is insensitive to the individual load each class imposes on the server and insensitive to the overall load the server experiences. This policy is known as the $c\mu$ -rule.

Recently and to address the fairness issue, some authors proposed that convex costs are a better way to model the problem. Customers in the non priority queues have a highly variable cycle time as well as they may have a long average waiting time, under linear costs. This may result in customer dissatisfaction and/or high abandonment rates. The policy derived for convex costs is shown to be asymptotically optimal for heavy traffic and consists on a generalization of the optimal policy for linear costs, taking the derivative of the single stage cost function. One of the characteristics preserved in the generalization is the insensitivity to the individual loads of the classes being served.

We claim and show that for convex costs the optimal policy depends on the individual loads. Therefore, there is a need for an alternative generalization of the $c\mu$ -rule. The main feature of our generalization consists on first order differences of the single stage cost function, rather than on its derivatives. The resulting policy is able to reach near optimal performances and is a function of the individual loads.

Keywords Scheduling · Production Control · Queuing Systems · Dynamic Priorities · $c\mu$ -rule

This work was supported by the FCT (ISR/IST plurianual funding) through the PIDDAC Program funds.

Carlos F. Bispo
Instituto de Sistemas e Robótica – Instituto Superior Técnico
Av. Rovisco Pais, 1049-001 Lisboa, Portugal E-mail: cfb@isr.ist.utl.pt

1 Introduction

The setting for the problem we address consists of a single server that can process different classes of customers, which arrive from the outside world and queue up in front of it, waiting for service. There will be one queue per class and each queue is served on a first come first serve basis. The arrival process is non controllable and each class requires different processing times that, in general, are assumed random and *a priori* unknown to the server. Whenever the server concludes a service, it will have to decide which of the classes to serve next, out of the ones which have customers present. It is assumed that there is a cost associated with each queue that is proportional to the number of customers on that queue or, conversely, proportional to the waiting time of the customers. So, the growing of the queues constitutes the incentive for the server to work.

In general, considering there are $x_i(t)$ customers of class i , for $i = 1, 2, \dots, K$, at the time instant t , the single stage cost for class i can be defined to be $C_i : N \rightarrow R$, such that $C_i(0) = 0$ and $\lim_{x_i \rightarrow \infty} C_i(x_i) = \infty$. Here we dropped the explicit time dependence for convenience. Furthermore, we will be interested in the cases where these functions are convex. For a finite time problem under some decision policy one may take the expected value over all possible trajectories of the integral over time of the single stage cost functions sum for all classes. If the length of the trajectory is unbounded one may choose to take a series of fixed length time average of that expected integral, with growing length – infinite horizon average costs –, or take the expected integral of the discounted sum of the single stage costs – infinite horizon discounted costs. An optimal policy will be the set of decisions, function of the k -tuple $(x_1, x_2, \dots, x_k)$ at any decision point, that for each of the above cases will achieve the minimal cost.

If the service can be preempted and later resumed for any class, there will be two decision points: the completion of a service and the occurrence of an arrival. For the non preemptive case, only a service conclusion is a decision point. The only exception to this is when an arrival occurs while the server is idle because, prior to the arrival and upon conclusion of the last service, there were no customers waiting for service. We also focus on the case where there is no cost associated with activating the server – no warm up cost –, and no cost associated with switching from a class to another – no set up, or change over, cost.

The classic approach to this problem assumed that the single stage costs are linear, e.g., $C_i(x_i) = c_i x_i$. When such is the case, for both infinite cost versions in general, the optimal policy is known as the $c\mu$ -rule. That is, assuming the average processing time for class i is $1/\mu_i$, the optimal policy is such that at any decision point the server will engage service with the head of the queue of the class which possesses the highest value of $\mu_i c_i$. An easy way to interpret intuitively this rule is to consider that, if all the processing rates are the same, priority should be given to the most costly queue, or to consider that, if all the single stage costs are the same, priority should be awarded to the queue with the shortest average processing time.

The oldest known reference to a version of this problem dates back to 1956. It is considered that Smith [15] was the first to suggest the optimality of the $c\mu$ -rule. His setting was deterministic and static. That is, the processing times are fixed for each class – deterministic –, and all the customers are present at time 0 and no arrivals are allowed after that – static. Outside the queuing theory community, in the scheduling theory community this is also referred to as the WSPT – Weighted Shortest Processing

Time – rule. Later, Cox and Smith [6] showed the $c\mu$ -rule to be optimal for a stochastic, dynamic environment with arbitrary time horizon. Their setting was that of a multiclass M/G/1 queue. They considered both preemptive and non preemptive cases. Naturally, it came with not much surprise that this rule is also optimal for stochastic and static settings as Pinedo [12] and Righter [14] are examples where such result can be found.

The amount of extensions and variants of the problem that have been considered after Cox and Smith is quite significant. For more references on related work we refer the reader to the literature review presented in [16]. We only consider a sample of the ones that focus on the simpler problem, i.e., no feedback for instance. Out of those, Harrison [9] considered a multiclass M/G/1 with the added feature that there are also rewards for each service completion. His policy is slightly more complex than the $c\mu$ -rule, as his β -optimal, β being the discount parameter on a continuous time problem, specify a priority ranking also, but some classes may never be served. The ranking is a function of β , which is not the case of the original problem. Also, the ranking is not defined by the simple $c\mu$ -rule. We believe that these differences are explained by the inclusion of rewards, which distorts the original problem significantly. For the case of discrete time problems, one example of optimality of the $c\mu$ -rule was presented by Buyukkoc, Varaiya, and Walrand [4], for multiclass systems under arbitrary arrival processes, geometric service times, and preemptive discipline. This followed the work of Baras, Dorsey, and Makowsky [2] which established the optimality of the $c\mu$ -rule, considering only two classes of customers, with arbitrary arrival processes and service completions generated by independent Bernoulli streams.

One of the most intriguing features of this problem is the fact that the arrival rates play no role on the optimal policy structure. Defining as λ_i the average arrival rate for class i and defining as individual load of class i the ratio λ_i/μ_i , the fact that the each class may have a higher or lower individual load is of no consequence on the optimal policy. This raises an issue of fairness, in some sense. Suppose there are only two classes of customers and the lower priority class has an individual load close to 10%, while the high priority class has an individual load close to 90% (heavy traffic), for instance costs are similar but processing rates are different. A customer of the non priority class will see many customers of the priority class being served first despite the fact that they may have arrived after. A consequence of this is a high variance of the waiting time for the non priority customers. Naturally, the non priority customers have more difficulty estimating when will they leave the system and, while waiting, each arrival they see occurring for the priority queue has to be a source of disappointment. The linear cost model tells us that the marginal patience of the customers is always the same no matter how much time they have been waiting. That is, the willingness to wait an extra time unit is the same after a handful of services that it was upon arrival. Whoever has stood in a non priority queue knows that this is not true and a natural consequence is a high abandonment rate, or worse. Naturally, one could argue that the abandonment behavior could be incorporated in the model and the appropriate policy could be, afterwards, derived. This is the case addressed by Harrison and Zeevi, [10].

Another interesting feature of the problem is the fact that the optimal policy is intrinsically myopic. That is, what appears to be the best short term decision agrees with the long term best decision. Associated with this, given its simple structure, it appears that the processing rate should be multiplied by the derivative of the cost function when the single stage costs are linear.

The first work on this problem that addresses the concern of fairness is that of Van Mieghem [16]. There, the author considers the single stage cost to be a convex function

of the delay for multiclass single server systems. Then, proposes to use the generalized $c\mu$ -rule, where c is replaced by the first derivative with respect to delay of the single stage cost function. By performing a heavy traffic analysis, the author shows that this generalized rule is asymptotically optimal, in the sense that the cost achieved under this policy approaches the cost of the optimal policy, as the sum of the individual loads approaches unity. Following this work, Mandelbaum and Stolyar [11] extend the analysis to a case where the single server is replaced by a pool multiskilled servers that work in parallel, considering convex single stage costs as functions of the individual queue lengths. They also establish the asymptotic optimality of the generalized $c\mu$ -rule by means of conducting a heavy traffic analysis. The maximum pressure policies of [7, 8] for general stochastic processing networks produce exactly Van Mieghem's generalized $c\mu$ -rule for single server problems.

While agreeing with the inclusion of convex costs to better reflect the marginal patience of the waiting customers, we believe that there are two points on the generalization that deserve further discussion. The first point concerns using the derivative of the single stage cost function to generalize the $c\mu$ -rule. Firstly, we stress that each $C_i : N \rightarrow R$ and one can construct many convex such functions which have no derivative when assuming their domain to be the set of the real numbers. Secondly, and probably the most relevant issue, suppose one formulates this as a continuous time Markov Decision Problem, assuming Poisson arrivals, exponentially distributed service times, and applies Dynamic Programming to compute the optimal policy, through a policy iteration algorithm, for instance. Given the fact that the state space is a k -tuple of integers and that through its successive iterations the algorithm only produces valid state space transitions, one should wonder how would it be possible to converge to derivatives. In other words, is the simplicity of the linear costs hiding something else?

The second point concerns the fact that the individual loads are still not playing any role on the structure of the optimal or sub-optimal policies, which is intriguing, to say the least. One exception to this is the work of [1], where the authors derive an index heuristic for convex costs, by formulating a restless-bandit problem. Their approach considers preemptive service only, and the resulting index is a function of the individual arrival rates, but only considers the cost gain of reducing the queue length of the served class.

It is the purpose of the work presented here to further our knowledge on this problem and to accomplish this we will show that the optimal policy does depend on the individual loads and that a better generalization of the $c\mu$ -rule relies on first order differences of the single stage cost function. Our generalization includes also the influence of cost increases when a queue gets an extra customer, not just the cost reduction due to a departing customer, as in [1]. On this last finding, note that for linear costs they are exactly the same, thus justifying that the linear costs do hide a more interesting feature.

Naturally, these findings will have to be reconciled with [16,11,8], as our work does not question the validity of the results there reported. In fact, the asymptotical optimality of their generalized $c\mu$ -rule, which we will term as the $Gc\mu$ -rule, does not conflict with the fact that, in general, we get costs no further from the optimal costs with our proposed sub-optimal policies and even achieve better results than the $Gc\mu$ -rule.

In what follows we will first formulate a MDP for a two class single server with convex single stage costs in Section 2. The restriction to two classes is done due to the fact that we intend to numerically compute the optimal policy and do not want to be

overwhelmed by the curse of dimensionality, [3]. Then, in Section 3, we establish a set of very interesting results for particular instances of the single server scheduling problem, that will help us identifying how should the $c\mu$ -rule be generalized. These results are valid *per se*, as some of the systems considered can occur in real life. Following this, in Section 4, we present a series of numerical examples that illustrate that the optimal policy is a function of the individual loads. Inspired by the results of Sections 3, we propose a generalization of the $c\mu$ -rule and present numerical data to support our claim that it is possible to have a better generalization than the existing ones. Finally, we conclude in Section 5, establishing a bridge between our work and previous work, and pointing directions for further research.

2 The model

We are going to restrict our analysis to a system serving only two classes of customers. Let λ_i be the average arrival rate for class i , for $i = 1, 2$, and assume customers arrive according to independent Poisson processes. The processing requirements of each customer are assumed to be statistically the same within each class, with service times being exponentially distributed with mean $1/\mu_i$. Each service duration is independent of previous service durations as well as independent of the number of customers waiting in the system. Once started, a service may or may not be preempted and later resumed with no penalty. We will address both cases where preemption is and is not allowed, because there are some issues worth discussion concerning the later. Upon conclusion of a service, the customer being served leaves the system.

We define as $X(t) = [x_1(t) \ x_2(t)]'$ the amount of customers of both classes present in the system at time t . Given that there is only one server, it may be the case that either a customer of class 1 or of class 2 is being served when $X(t)$ is in the positive quadrant, while all the others are waiting. Also, we assume idleness as a possible decision for the server, although it will be seen later that the server never chooses to remain idle if there is at least one customer in one of the two queues. Given the fact that customers in the same queue are undistinguishable, each queue is served by the order of their arrival to the system, although customers of a given queue may be served prior to customers of the other queue that arrived earlier to the system.

Our state description will also have to include the state of the server when we consider the no preemption model. Therefore, we define as $Z(t) = [X'(t) \ y(t)]'$ the state of the system, where $y(t) \in \{0, 1, 2\}$ is the server state at time t . If $y(t) = 0$ the server is idle, or serving a customers of class i , if $y(t) = i$.

We will consider an infinite horizon discounted cost criterion with discount parameter $\beta > 0$ and will be interested in obtaining a stationary Markov policy. That is, a policy is defined as a function that maps the state into one of the three options for the server state, such that the decision is not a function of the time instant.

With the instantaneous cost rate defined earlier we can define the expected present value of future costs, under a policy π , as follows

$$J(Z(0), \pi) = E_{Z(0)}^\pi \left\{ \int_0^\infty e^{-\beta t} C(Z(t)) dt \right\}, \quad (1)$$

where $E_{Z(0)}^\pi \{.\}$ denotes the expectation with respect to the probability distribution of the path space of Z that corresponds to initial state $Z(0)$ and control policy π , and $C(Z(t)) = C_1(x_1(t)) + C_2(x_2(t))$. We then define the value function as

$$V(Z(0)) = \inf_{\pi \in \Pi} J(Z(0), \pi) \quad \text{for } Z(0) \in S, \quad (2)$$

where S defines the set of all possible states and Π defines the set of all stationary policies. We will use $V(X(0))$ in the preemptive case and $V(X(0), y(0))$ in the non preemptive case. In what follows we will first detail the preemptive case followed by the detail of the non preemptive case. After we compare the equations and show that the standard formulation of MDP needs to be changed to accommodate systems where the server state need to be captured in the overall state description.

2.1 Detail for the non preemptive case

When preemption is not allowed, the only decision points are arrivals to an empty system or conclusion of service. So, only for $y = 0$ we have choices to make. Because we are dealing with a continuous time Markov process, we resort to the uniformization procedure to convert it into a discrete time problem. Defining the uniform rate as $\gamma \geq \lambda_1 + \lambda_2 + \mu_1 + \mu_2 \geq 0$, $\alpha = \gamma/(\beta + \gamma)$, and omitting the explicit time dependency to avoid an excessively cumbersome notation, the value iteration algorithm, [3], for this problem becomes

$$V_{k+1}(X, 0) = \frac{1}{\beta + \gamma} [C_1(x_1) + C_2(x_2)] + \alpha \min \{ \tilde{V}_k(X, 0, u|u=0), \tilde{V}_k(X, 0, u|u=1), \tilde{V}_k(X, 0, u|u=2) \}, \quad (3)$$

where u represents the control decision, $V_0(X) = 0, \forall X \in S$, and with

$$\begin{aligned} \tilde{V}_k(X, 0, u|u=0) &= \frac{\lambda_1}{\gamma} V_k(X + e_1, 0) + \frac{\lambda_2}{\gamma} V_k(X + e_2, 0) + (1 - \frac{\lambda_1 + \lambda_2}{\gamma}) V_k(X, 0), \\ \tilde{V}_k(X, 0, u|u=1) &= \frac{\lambda_1}{\gamma} V_k(X + e_1, 1) + \frac{\lambda_2}{\gamma} V_k(X + e_2, 1) + \frac{\mu_1}{\gamma} V_k(X - e_1, 0) + \\ &\quad + (1 - \frac{\lambda_1 + \lambda_2 + \mu_1}{\gamma}) V_k(X, 1), \\ \tilde{V}_k(X, 0, u|u=2) &= \frac{\lambda_1}{\gamma} V_k(X + e_1, 2) + \frac{\lambda_2}{\gamma} V_k(X + e_2, 2) + \frac{\mu_2}{\gamma} V_k(X - e_2, 0) + \\ &\quad + (1 - \frac{\lambda_1 + \lambda_2 + \mu_2}{\gamma}) V_k(X, 2), \end{aligned} \quad (4)$$

with $e_1 = [1; 0]'$ and $e_2 = [0; 1]'$. To complete the model we need to present the operator for the remaining states.

$$V_{k+1}(X, 1) = \frac{1}{\beta + \gamma} [C_1(x_1) + C_2(x_2)] + \alpha \left\{ \frac{\lambda_1}{\gamma} V_k(X + e_1, 1) + \frac{\lambda_2}{\gamma} V_k(X + e_2, 1) + \frac{\mu_1}{\gamma} V_k(X - e_1, 0) + (1 - \frac{\lambda_1 + \lambda_2 + \mu_1}{\gamma}) V_k(X, 1) \right\}, \quad (5)$$

$$V_{k+1}(X, 2) = \frac{1}{\beta + \gamma} [C_1(x_1) + C_2(x_2)] + \alpha \left\{ \frac{\lambda_1}{\gamma} V_k(X + e_1, 2) + \frac{\lambda_2}{\gamma} V_k(X + e_2, 2) + \frac{\mu_2}{\gamma} V_k(X - e_2, 0) + \left(1 - \frac{\lambda_1 + \lambda_2 + \mu_2}{\gamma}\right) V_k(X, 2) \right\}. \quad (6)$$

Naturally, (5) is only applicable for states where $x_1 > 0$ and (6) for states where $x_2 > 0$.

3 Exact results on specific systems

In an effort to better understand the nature of the optimal policy for the problem addressed in this paper, we are now going to analyze four particular problems that have some connection with it. We start by defining the problems in a somewhat loose manner.

Problem 1 Take a static version of the problem addressed in this paper, with $K > 1$ classes. That is, all customers are present at time zero and no arrivals will occur afterward. Assume there are x_i customers in queue i and that the single stage cost is convex as defined earlier. The objective is to clear the system of customers with the lowest cost possible.

Problem 2 Consider a closed queuing network with a single server, two classes of customers, and fixed population. At the conclusion of a service on a given class a new customer of the other class is allowed to enter. Initially there are x_i customers of class i . With the same single stage cost as defined earlier, the objective is to identify the non idling policy that minimizes the infinite horizon discounted cost.

Problem 3 Consider a closed queuing network with a single server, $K = 2$ classes of customers, and fixed population. At the conclusion of a service on any given class a new customer will be allowed to enter the system. The new customer is of class i according to the ratio $p_i = \lambda_i / (\sum_{k=1}^K \lambda_k)$. With the single state cost defined earlier and x_i customers of class i present in the system at time zero, the objective is to identify the non idling policy that minimizes the infinite horizon discounted cost.

Problem 4 Consider an open queuing network with a single server and two classes of customers. At the conclusion of a service one customer of each class is allowed to enter the system. Assuming there are x_i customers of class i at time zero and using the single stage cost defined earlier, the objective is to identify the non idling policy that ensures the minimum infinite horizon discounted cost.

Before analyzing each of the four problems individually we offer some remarks on each problem. The first remark on the four problems is the fact that the arrival process is no longer uncontrollable. Naturally, knowing that no customers will arrive or that they only arrive when a service is concluded drastically changes the nature of the problem. An intrinsic feature of the single server scheduling problem we are addressing is the fact that only the stochastic nature of the arrival process is known, not the specific arrival instants.

Problem 1 can be seen as the convex cost successor, with stochastic services, of the original problem addressed by Smith, [15]. Also, in many service contexts there is such a thing as the closing hours, after which only the customers already inside the system will be served. At that point in time, when the doors are closed, the problem to be solved no longer is an infinite horizon dynamic problem, becoming static as Problem 1. Problems 2 and 3 are examples of manufacturing contexts where there is a fixed number of pallets where parts are mounted on for processing. So, only when a part is completed another one will use the available pallet. Problem 4 is naturally the oddest of them all, given the fact that it is unstable, whereas Problems 2 and 3 are marginally stable. Therefore, the concept of minimal cost needs to be clarified here. No matter what customer is served, two new customers will enter the system. Therefore, for any non idling policy chosen, the population will grow to infinity. We are looking for the non idling policy that achieves the infimum cost relative to all possible non idling policies. In other words, the policy that approaches infinity the cheapest way. Although this problem has no real life application, we hope its usefulness for our discussion will become clear by the end of this section. For the four problems we are able to characterize the structure of the optimal policy.

Lemma 1 In a situation where there are no arrivals during service, either because arrivals are switched off or because they only occur at the conclusion of a service, and assuming the first and second services will serve different queues, the value function for a given policy π can be written as

$$J(Z(0), \pi) = \frac{C(Z(0))}{\mu_i + \beta} + \frac{E_{Z(0)}^\pi \{C(Z(s_1))\}}{(\mu_i + \beta)(\mu_j + \beta)} \mu_i + E_{Z(0)}^\pi \left\{ \int_{s_1+s_2}^{\infty} e^{-\beta t} C(Z(t)) dt \right\}, \quad (7)$$

where $Z(0)$ is the initial state, μ_i is the processing rate of the first class served, μ_j is the processing rate of the second class served, s_1 is the duration of the first service, and s_2 the duration of the second service.

Proof: First, note that (1) can be written as

$$\begin{aligned} J(Z(0), \pi) &= E_{Z(0)}^\pi \left\{ \int_0^{s_1} e^{-\beta t} C(Z(t)) dt + \int_{s_1}^{s_1+s_2} e^{-\beta t} C(Z(t)) dt + \right. \\ &\quad \left. + \int_{s_1+s_2}^{\infty} e^{-\beta t} C(Z(t)) dt \right\} \\ &= E_{[Z(0), s_1]} \left\{ \int_0^{s_1} e^{-\beta t} C(Z(t)) dt \right\} + \\ &\quad + E_{[Z(0), s_1, s_2]} \left\{ \int_{s_1}^{s_1+s_2} e^{-\beta t} C(Z(t)) dt \right\} + \\ &\quad + E_{Z(0)}^\pi \left\{ \int_{s_1+s_2}^{\infty} e^{-\beta t} C(Z(t)) dt \right\} \end{aligned} \quad (8)$$

where s_1 is the duration of the first service and s_2 is the duration of the second service. We will now derive the expressions for the first two terms.

$$\begin{aligned}
 A_1 &= E_{[Z(0), s_1]} \left\{ \int_0^{s_1} e^{-\beta t} C(Z(t)) dt \right\} \\
 &= \int_0^\infty \int_0^{s_1} e^{-\beta t} C(Z(t)) dt \mu_i e^{-\mu_i s_1} ds_1 \\
 &= C(Z(0)) \int_0^\infty \int_0^{s_1} e^{-\beta t} dt \mu_i e^{-\mu_i s_1} ds_1 \\
 &= \frac{C(Z(0))}{\mu_i + \beta}.
 \end{aligned} \tag{9}$$

The above relation is valid if and only if there are no arrivals during service, which is the case of any of the four problems presented. Moreover, the number of customers on both queues equals those present at time zero, and the service on any of the queues is exponentially distributed.

For the second term, assuming the first and second services are on different classes and that there are no arrivals during the second service, we get

$$\begin{aligned}
 A_2 &= E_{[Z(0), s_1, s_2]} \left\{ \int_{s_1}^{s_1+s_2} e^{-\beta t} C(Z(t)) dt \right\} \\
 &= \int_0^\infty \int_0^\infty \int_{s_1}^{s_1+s_2} e^{-\beta t} C(Z(t)) dt \mu_i e^{-\mu_i s_1} ds_1 \mu_j e^{-\mu_j s_2} ds_2 \\
 &= E_{Z(0)}^\pi \{C(Z(s_1))\} \int_0^\infty \int_0^\infty \int_{s_1}^{s_1+s_2} e^{-\beta t} dt \mu_i e^{-\mu_i s_1} ds_1 \mu_j e^{-\mu_j s_2} ds_2 \\
 &= \frac{E_{Z(0)}^\pi \{C(Z(s_1))\}}{(\mu_i + \beta)(\mu_j + \beta)} \mu_i
 \end{aligned} \tag{10}$$

Q.E.D.

Definition 1 Let the first order difference of the single stage cost function along direction i at state Z be defined as $\Delta_i(x_i) = C(Z) - C(Z - e_i) = C_i(x_i) - C_i(x_i - 1)$.

Now we are in a position to characterize the optimal policies for the four problems.

Theorem 1 For Problem 1, when there are x_i customers in queue i , for $i = 1, \dots, K$, it is optimal to select for service the class for which $\mu_i \Delta_i(x_i)$ is maximum.

Proof: We use a pairwise interchange argument. Assume the optimal policy, π , is such that the k^{th} decision choses class j and the $(k+1)^{th}$ decision choses class i and the condition of the theorem is violated. That is, $\mu_j \Delta_j(x_j) < \mu_i \Delta_i(x_i)$. We build an alternative policy π' where only those two decisions are altered, serving first class i followed by class j .

Under both policies, all decisions taken prior to the k^{th} decision and after the $(k+1)^{th}$ decision will incur the same cost. Given the nature of the problem, we can assume with no loss of generality that $k = 1$.

So, we can compare the costs of serving first class j followed by class i with the costs of serving first class i followed by class j , and the following decisions will follow the optimal policy π .

For policy $\pi' = [i, j, \pi]$ and from Lemma 1 we get

$$J(Z(0), \pi') = \frac{C(Z(0))}{\mu_i + \beta} + \frac{C(Z(0) - e_i)}{(\mu_i + \beta)(\mu_j + \beta)} \mu_i + A(Z(0) - e_i - e_j, \pi'). \quad (11)$$

The term $A(Z(0) - e_i - e_j, \pi')$ represents the cost to go after the second service is concluded and policy π is used from then on, taking into account that a class i customer was served followed by a class j customer in the first two services.

For policy $\pi = [j, i, \pi]$ we get

$$J(Z(0), \pi) = \frac{C(Z(0))}{\mu_j + \beta} + \frac{C(Z(0) - e_j)}{(\mu_j + \beta)(\mu_i + \beta)} \mu_j + A(Z(0) - e_i - e_j, \pi), \quad (12)$$

By the nature of the problem and given the stated policies, $A(Z(0) - e_i - e_j, \pi') = A(Z(0) - e_i - e_j, \pi)$, because the state reached after the first two decisions is the same under both policies and policy π' produces the same decisions as policy π from the third decision onward. Defining $\Delta_J = J(Z(0), \pi') - J(Z(0), \pi)$

$$\begin{aligned} \Delta_J &= \frac{C(Z(0))}{\mu_i + \beta} + \frac{C(Z(0) - e_i)}{(\mu_i + \beta)(\mu_j + \beta)} \mu_i - \frac{C(Z(0))}{\mu_j + \beta} - \frac{C(Z(0) - e_j)}{(\mu_i + \beta)(\mu_j + \beta)} \mu_j \\ &= \frac{1}{(\mu_i + \beta)(\mu_j + \beta)} \{ C(Z(0))(\mu_j - \mu_i) + C(Z(0) - e_i)\mu_i - C(Z(0) - e_j)\mu_j \} \\ &= \frac{1}{(\mu_i + \beta)(\mu_j + \beta)} \{ -C(Z(0))\mu_i + C(Z(0) - e_i)\mu_i + C(Z(0))\mu_j - \\ &\quad - C(Z(0) - e_j)\mu_j \} \\ &= \frac{1}{(\mu_i + \beta)(\mu_j + \beta)} \{ -\Delta_i(x_i)\mu_i + \Delta_j(x_j)\mu_j \}. \end{aligned} \quad (13)$$

Given the fact that, $\frac{1}{(\mu_i + \beta)(\mu_j + \beta)} > 0$ and $\mu_j \Delta_j(x_j) < \mu_i \Delta_i(x_i)$, it follows that $\Delta_J < 0$, which contradicts the optimality assumption for policy π .

We can apply the same argument for all consecutive decisions where different classes are served. Therefore, from the optimal policy π we can construct an alternative policy, π^* , where costs are never worse than those achieved under policy π by enforcing the stated rule whenever it is not observed in policy π , and the result follows.

Q.E.D.

The formal results for the remaining problems will be presented without proofs, as the proof technique follows the same path as this one.

Theorem 2 For Problem 2, with x_i customers in queue i , for $i = 1, 2$, it is optimal to select for service class i if $\mu_i \Delta_i(x_i) + \mu_j \Delta_i(x_i + 1)$ is maximum, where j is the index for the other class.

Theorem 3 For Problem 3, with x_i customers in queue i , for $i = 1, 2$, and defining $p_i = \lambda_i / \sum_k \lambda_k$, it is optimal to select for service the class i if $p_j \mu_i \Delta_i(x_i) + p_i \mu_j \Delta_i(x_i + 1)$ is maximum, where j is the index for the other class.

Theorem 4 For Problem 4, with x_i customers in queue i , for $i = 1, 2$, it is optimal to select for service the class i if $\mu_j \Delta_i(x_i + 1)$ is maximum, where j is the index for the other class.

Before concluding this section, the solution for Problem 4 deserves some intuitive description as it can be seen as a dual to the solution for Problem 1. For Problem 4, if both service rates are the same we should chose to serve the class that most increases cost by receiving the extra customer. If both cost increases are the same we should chose to serve the class with the lowest processing rate to delay the cost increase due to the extra customer.

The relevance of Problem 4 to our discussion should now be obvious. For any of the problems, we use a combination of the term $\mu_i \Delta_i(x_i)$ with the term $\mu_j \Delta_i(x_i + 1)$. Apart Problem 2, the other ones are covered by the whole set of convex combinations of the two first order differences, when there are only two classes. Table 1 contains the synthesis of the results just established. The first remark we need to make concerns the fact that any choice of p_i and p_j such that each is non negative and $p_i + p_j = 1$ will result on the optimal policy for one instance of the above problems with only two classes. If both multipliers are equal to 1, then we are producing the optimal policy for Problem 2. Therefore, if the optimal policy for the problem we address in this paper was also to be produced by means of a combination of first order differences, the multipliers to be used would have to be such that one or both would take values outside the $[0; 1]$ interval.

Table 1 The optimal multipliers.

Problem	$\mu_i \Delta_i(x_i)$	$\mu_j \Delta_i(x_i + 1)$
1	1	0
2	1	1
3	p_j	p_i
4	0	1

Note that we are able to completely specify the optimal policies by means of a combination of first orders differences of the single stage cost function, which can be easily computed, and the optimal policies do not depend on the discount parameter being used. It is also relatively easy to prove the above results for the infinite horizon average costs. The proof scheme for these results is also based on a pairwise interchange argument. We do not present those to avoid redundancy of the proofs, and expect any interested reader to be able to produce them, as they are simpler because there is no discount term to clutter the equations.

Finally, note that the convexity assumption for the single stage cost function does include the case of linear costs, meaning the above results to be valid also for those. Also, the above results are trivially valid for the preemptive and the non preemptive cases. Since we know that the optimal policy for linear costs is the $c\mu$ -rule, it would appear that the problem with linear costs is equivalent to our Problem 1. In a sense it is, because the arrivals during service are irrelevant in a pairwise interchange argument, as can be seen in Cassandra, [5], pp. 492—495, for the discrete time version of the problem. However, it is the claim of this paper that there is a little more to it, as we will show in the following sections.

4 Toward a generalization of the $c\mu$ -rule

In this section we present numeric evidence of the fact that the optimal policy for the convex cost version of the scheduling problem depends on the individual load. Given that we assume the buffers to be unbounded and we have to run the value iteration algorithm for bounded state space we ran a series of tests to define an acceptable cutting point, so that the value of the optimal cost is as close as possible equal to the value for unbounded state space. For all the systems we run the value iteration algorithm with $x_i \in [0; 200]$ and make $\beta = 0.001$. For results presentation we further cut the state space where we can be sure there are no errors due to the approximation. The results will be presented for $x_i \in [0; 50]$. After these results we will propose an alternative generalization of the $c\mu$ -rule, discuss its structure, and will present numeric evidence supporting its adequacy in terms of performance.

System 1 Consider a non preemptive system with $C_1(x_1) = 2x_1$, $C_2(x_2) = 1.001x_2 + 0.1x_2^2$, $\mu_1 = 2$, and $\mu_2 = 1$.

Mieghem's generalized $c\mu$ -rule, which we will refer to as the $Gc\mu$ -rule, will produce the following condition.

$$u(x_1, x_2) = \begin{cases} 0 & \text{if } x_i = 0 \\ 1 & \text{if } x_1 > 0 \wedge x_2 \leq 14.995 \\ 2 & \text{if } x_2 > 0 \wedge (x_1 = 0 \vee x_2 > 14.995), \end{cases}$$

which is a threshold policy on the value of x_2 . Whenever x_2 is below 15, priority is given to class 1. If x_2 is above 14, class 2 has priority over class 1. Note that this policy does not depend the $\rho_1 = \lambda_1/\mu_1$ and $\rho_2 = \lambda_2/\mu_2$, nor on $\rho = \rho_1 + \rho_2$.

If the quadratic term of the single stage cost of class 2 were zero, then the optimal policy would be to give priority to class 1 at all times. The existence of the quadratic term for class 2 forces the decision maker to change priority when the population of class 2 becomes higher than some amount. The higher the quadratic term, the lower will the threshold be.

If we compute the optimal policy, using the value iteration algorithm, we also get a threshold type policy as a function of x_2 . Table 2 presents the optimal threshold values for a sample of values of ρ_i and a global load of 70% and 90%.

Table 2 Optimal threshold values for System 1.

ρ_1	ρ_2	Threshold	
		$\rho = 70\%$	$\rho = 90\%$
0.05	$\rho - 0.05$	14	10
0.10	$\rho - 0.10$	14	12
$\rho/2$	$\rho/2$	15	15
$\rho - 0.10$	0.10	16	16
$\rho - 0.05$	0.05	16	16

When the individual loads are the same, the optimal threshold equals the one obtained by the $Gc\mu$ -rule. However, in the case of fixed global load, if the individual load of, say, class 2 grows, the optimal threshold decreases. This behavior is consistent

with intuition in the following sense. There should be a threshold for x_2 above which class 2 gets priority over class 1. If the traffic for class 2 is heavier, then the threshold should drop because there will be more customers of class 2. This threshold drop does not affect the performance for class 1, given that their input rate is lower and their queue does not get as many customers as often as queue 2 gets. Moreover, we see that as the global load increases, there is a tendency for the span of the optimal threshold to get wider as a function of the individual loads. In fact, the optimal threshold for a global load of 95% and for the first line of Table 2 is 7 and for last line is still 16.

System 2 Consider a non preemptive system with $C_1(x_1) = 1.001x_1 + 0.05x_1^2$, $C_2(x_2) = 5x_2$, $\mu_1 = 2$, and $\mu_2 = 1$.

Given that the quadratic term is present on the cost associated with class 1, the $Gc\mu$ -rule will produce a threshold policy for class 1, according to the following.

$$u(x_1, x_2) = \begin{cases} 0 & \text{if } x_i = 0 \\ 1 & \text{if } x_1 > 0 \wedge (x_1 \geq 14.99 \vee x_2 = 0) \\ 2 & \text{if } x_2 > 0 \wedge x_1 < 14.99. \end{cases}$$

Table 3 presents the threshold results for a global load of 90%. The behavior for this system follows the same principle as for System 1. However, for this system the threshold for equal loads does not match the level obtained through differentiation of the single stage cost function.

Table 3 Optimal threshold values for System 2.

ρ_1	ρ_2	Threshold
0.05	$\rho - 0.05$	16
0.10	$\rho - 0.10$	16
$\rho/2$	$\rho/2$	14
$\rho - 0.10$	0.10	10
$\rho - 0.05$	0.05	9

System 3 Consider a non preemptive system with $C_1(x_1) = x_1 + 0.1x_1^2$, $C_2(x_2) = 1.2x_2 + 0.3x_2^2$, $\mu_1 = 3$, and $\mu_2 = 1$.

Applying again the $Gc\mu$ -rule we get

$$u(x_1, x_2) = \begin{cases} 0 & \text{if } x_i = 0 \\ 1 & \text{if } x_1 > 0 \wedge (x_2 \leq x_1 + 3 \vee x_2 = 0) \\ 2 & \text{if } x_2 > 0 \wedge (x_1 = 0 \vee x_2 > x_1 + 3). \end{cases}$$

System 4 Consider a non preemptive system with $C_1(x_1) = x_1 + 0.1x_1^2$, $C_2(x_2) = 4.8x_2 + 0.3x_2^2$, $\mu_1 = 3$, and $\mu_2 = 1$.

For this system we get the following conditions for the $Gc\mu$ -rule.

$$u(x_1, x_2) = \begin{cases} 0 & \text{if } x_i = 0 \\ 1 & \text{if } x_1 > 0 \wedge (x_2 \leq x_1 - 3 \vee x_2 = 0) \\ 2 & \text{if } x_2 > 0 \wedge (x_1 = 0 \vee x_2 > x_1 - 3). \end{cases}$$

Figure 1 presents the optimal switching curves obtained for different individual loads when the global load is 90% for Systems 3 and 4. When the state of the system at a decision point is above or over any of the curves the server choses class 2 and below choses class 1. Each plot also includes the switching curve associated with the $Gc\mu$ -rule. The labels "5%" and "85%" correspond to the individual load of class 1. Taking the generalized curve as reference, we see that an increase of the individual load for class 1 shifts the curve upward, and a decrease in its individual load shifts the curve downward. This behavior is in line with what was shown for Systems 1 and 2. One could also say that the curves move left or right, given that the observed shifts are consistent with the both observations. Note that the slope of the optimal switching curves is the same as the slope of the generalized curve, which is 1. We need to clarify the little detail at the origin where we intentionally changed one of the curves to mark the square where the server is idle because there are no customers.

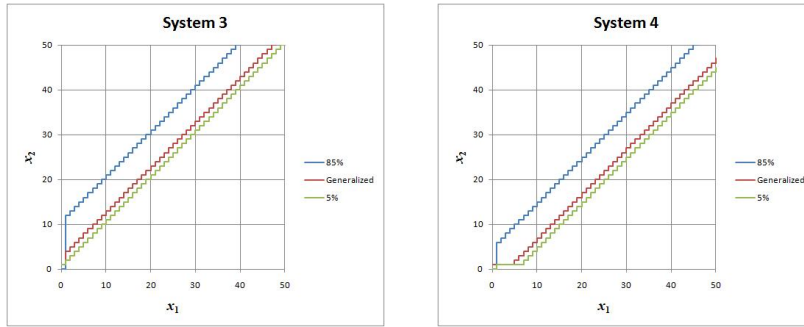


Fig. 1 Switching curves for Systems 3 and 4.

Taking these four systems as examples, one would be led to believe that the optimal switching curves correspond to some type of shift of the switching curve obtained by the $Gc\mu$ -rule. The following system serves the purpose of showing that it is not so.

System 5 Let there be a non preemptive system with $C_1(x_1) = 2^{-9}x_1^3$, $C_2(x_2) = 2^{-5}x_2^3$, $\mu_1 = 4$, and $\mu_2 = 1$.

Applying the $Gc\mu$ -rule we get the following

$$u(x_1, x_2) = \begin{cases} 0 & \text{if } x_i = 0 \\ 1 & \text{if } x_1 > 0 \wedge x_2 \leq 0.5x_1 \\ 2 & \text{if } x_2 > 0 \wedge (x_1 = 0 \vee x_2 > 0.5x_1), \end{cases}$$

The optimal switching curves for $\rho_1 = 0.05$ and $\rho_1 = 0.85$, when the global load is 90%, are displayed in Figure 2, together with the generalized curve. Observing closely the optimal curves for low values of x_i we see that they are not straight lines, although

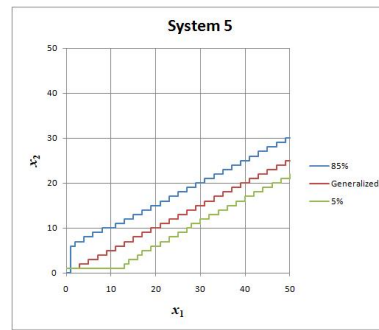


Fig. 2 Switching curves for System 5.

as $|X|$ grows they become straight lines parallel to the generalized curve. This numeric evidence on the structure of the optimal policy and its structural differences relative to the switching curve generated through taking the derivative of the single stage cost function calls for an attempt to propose a different manner of how to generalize the $c\mu$ -rule. Before we move on to propose a new generalization, we need to analyze the performance achieved under the optimal policies versus the performance achieved under the $Gc\mu$ -rule.

Table 4 presents $V(X,0)$ for $X = [0;0]'$ achieved with the optimal policy and with Miegheem's generalized rule, under "Derivative", for the first two systems. The global load is 90%. Although the threshold levels may be significantly different on both situations, as shown earlier, we see that the observed performance deviation is practically insignificant, agreeing with the claim of asymptotic optimality.

Table 4 Policy performance for Systems 1 and 2.

ρ_1	System 1			System 2		
	Optimal	Derivative	Deviation	Optimal	Derivative	Deviation
5%	20,371.2	20,530.4	0.781%	36,184.6	36,187.8	0.009%
10%	19,290.6	19,391.1	0.521%	34,076.4	34,079.3	0.009%
45%	15,119.5	15,119.5	0.000%	25,028.0	25,033.5	0.022%
80%	13,660.4	13,661.5	0.008%	17,429.7	17,563.7	0.769%
85%	14,500.0	14,500.4	0.003%	16,585.5	16,754.5	1.019%

A similar behavior is recorded for System 3 and 4, Table 5. However, for System 4 we note a maximum performance deviation on the sampled cases of over 5%. It appears that there would be no advantage in trying to improve performance, given that the results displayed so far are so favorable to the existing sub-optimal policy. However, turning again to System 5 we see in Table 6 that the performance deviation can be much higher. We recorded a little over 19% deviation in performance for this particular system, which is very significant.

Therefore, given the small sample of systems presented and the potential performance deviation that may occur, we believe there is room for improvement in an attempt to produce alternative approximations to the optimal policy for convex costs. Naturally, even if the performance deviations were not significant, the simple fact that the optimal policy is a function of the individual loads is in itself an interesting aspect

Table 5 Policy performance for Systems 3 and 4.

ρ_1	System 3			System 4		
	Optimal	Derivative	Deviation	Optimal	Derivative	Deviation
5%	43,697.5	43,895.7	0.454%	66,859.6	66,978.5	0.178%
45%	24,681.1	24,790.5	0.443%	39,656.2	39,816.6	0.404%
85%	14,500.0	14,500.4	0.003%	21,224.8	22,383.8	5.461%

of the problem we are addressing. This fact alone has escaped a long series of work in the area, for over half a century. Our challenge is to identify a better approximation of the optimal policy, that at the same time is compatible with the optimal policy for linear costs. In order to achieve that, we will build on the results of Section 3 and on the insights gained by the numeric examples of this section.

Table 6 Policy performance for System 5.

ρ_1	Optimal	Derivative	Deviation
5%	55,319.6	56,721.2	2.534%
45%	14,933.7	15,082.9	0.999%
85%	3,920.7	4,673.1	19.189%

4.1 Alternative generalization

The motivation that led to the generalized $c\mu$ -rule based on derivatives has been the coincidence between the fact that c_i is the derivative of cost for class i , assuming $C_i(x_i) = c_i x_i$. Also, the heavy traffic analysis is partially responsible for this. In fact, by scaling time and state space to convert a discrete problem into a continuous problem, the emergence of derivatives is a natural consequence.

We propose to alternatively consider a combination of first order differences, designated as the $\Delta c\mu$ -rule, according to the following.

$$\mu_i[q_i \Delta_i(x_i) + r_i \Delta_i(x_i + 1)], \quad (14)$$

where $q_i + r_i = 1$ and both are real numbers. The fact that $\Delta_i(x_i) = \Delta_i(x_i + 1) = c_i$ when costs are linear, ensures the desired compatibility with the $c\mu$ -rule. Recalling Table 1 we could re-write the expression above as

$$\mu_i[q_i \Delta_i(x_i) + \frac{\mu_j}{\mu_i} \hat{r}_i \Delta_i(x_i + 1)], \quad (15)$$

defining $r_i = \hat{r}_i \mu_j / \mu_i$, which leads to

$$\mu_i q_i \Delta_i(x_i) + \mu_j \hat{r}_i \Delta_i(x_i + 1). \quad (16)$$

With q_i taking the place of p_j and $\hat{r}_i$ taking the place of p_i in Table 1, at the end of Section 3 we observed that if $q_i + \hat{r}_i = 1$ and both parameters are in $[0; 1]$, we produce the optimal policy for some instance of Problems 1, 3, or 4. Then we claimed that to

address other problems one would have to lift those constraints. Saying that $q_i + r_i = 1$ is equivalent to saying that $q_i + \hat{r}_i \mu_j / \mu_i = 1$, which in general means that $q_i + \hat{r}_i \neq 1$.

Given the fact that the optimal policy depends on the individual loads, both q_i and r_i will have to depend on the individual loads too. That is, we need to write $q_i(\rho_1, \rho_2)$ and $r_i(\rho_1, \rho_2)$ in general. We omitted that dependence earlier to simplify the presentation. We will now analyze the structure of the switching curves produced by this scheme for general systems.

Consider a system where $C_1(x_1) = c_{11}x_1$ and $C_2(x_2) = c_{21}x_2 + c_{22}x_2^2$. The equation defining the switching curve is given by

$$\begin{aligned} \mu_1 c_{11} &= \mu_2 [q_2(c_{21} + 2c_{22}x_2 - c_{22}) + r_2(c_{21} + 2c_{22}x_2 + c_{22})] \\ &= \mu_2 [c_{21} + 2c_{22}x_2 + c_{22}(r_2 - q_2)], \end{aligned} \quad (17)$$

because $\Delta_1(x_1) = \Delta_1(x_1 + 1) = c_{11}$ and taking arbitrary values for q_i and r_i , such that $q_i + r_i = 1$. Therefore, we can re-write (17) to explicitly account for the threshold level of x_2 , as follows.

$$x_2 = \frac{\mu_1 c_{11} - \mu_2 c_{21}}{2\mu_2 c_{22}} - \frac{r_2 - q_2}{2}. \quad (18)$$

If $q_2 = r_2$ we get the same threshold as the $Gc\mu$ -rule. If $q_2 < r_2$, the threshold will go down and it will go up when $q_2 > r_2$. The essential feature we want to stress here is the fact that this formulation produces a threshold that can be shifted up or down, depending on the individual loads. To achieve the range displayed in Table 2 we need to make $q_2 < 0$ and $q_i > 1$ when ρ_1 is low. More specifically, if $r_2 - q_2 = 9.99$, we achieve a threshold of 10 for x_2 . With the assumption that $q_2 + r_2 = 1$ this is accomplished with $r_2 = 5.495$ and $q_2 = -4.495$.

Now assume we have a system such that $C_1(x_1) = c_{11}x_1 + c_{12}x_1^2$ and $C_1(x_1) = c_{11}x_1 + c_{12}x_1^2$. Using the $\Delta c\mu$ -rule, the switching curve will be given by

$$\mu_1 [c_{11} + 2c_{12}x_1 + c_{12}(r_1 - q_1)] = \mu_2 [c_{21} + 2c_{22}x_2 + c_{22}(r_2 - q_2)], \quad (19)$$

rearranging the terms and solving in order to x_2 , we get

$$x_2 = \frac{\mu_1 c_{12}}{\mu_2 c_{22}} x_1 + \frac{\mu_1 c_{11} - \mu_2 c_{22}}{2\mu_2 c_{22}} + \frac{\mu_1 c_{12}(r_1 - q_1) - \mu_2 c_{22}(r_2 - q_2)}{2\mu_2 c_{22}},$$

which is a family of straight lines with constant slope and variable intercept. The intercept variation is introduced by the parameters q_i and r_i . Again, if $q_i = r_i$ we get the equation generated by the $Gc\mu$ -rule. This structure is consistent with what we observed for Systems 3 and 4. For systems like System 1 and 2, knowing the value of the threshold is enough to determine the values of the two necessary parameters. However, for systems like System 3 and 4, knowing the intercept is not enough to determine the values of the four parameters, as we are left with one more degree of freedom. As we will show in the following and much to our surprise, the existence of the extra degree of freedom seems to be apparent.

Finally, assuming pure cubic costs for both classes, that is, $C_i(x_i) = c_{i3}x_i^3$, we get

$$\mu_1 [3c_{13}x_1^2 + 3c_{13}x_1(r_1 - q_1) + 1] = \mu_2 [3c_{23}x_2^2 + 3c_{23}x_2(r_2 - q_2) + 1], \quad (20)$$

which can easily be solved in order to x_2 . Inspecting this last expression, when $|X| \rightarrow \infty$ the quadratic terms are dominant. Thus the curve approaches a straight line. However, for low values of $|X|$ the equation above exhibits a non linear behavior. This is in line with what was observed in Figure 2 for the optimal switching curves displayed.

4.2 Numeric results

Once we have proposed a new generalization for the $c\mu$ -rule, it is necessary to evaluate how close to the optimal performance is it possible to get by tuning the parameters introduced previously. We will focus our analysis on System 5 exclusively. First, we will take a fixed value for ρ_1 and ρ_2 and will show how the performance depends on the parameters. After, we will present a map of the best achieved performances for a range of individual loads.

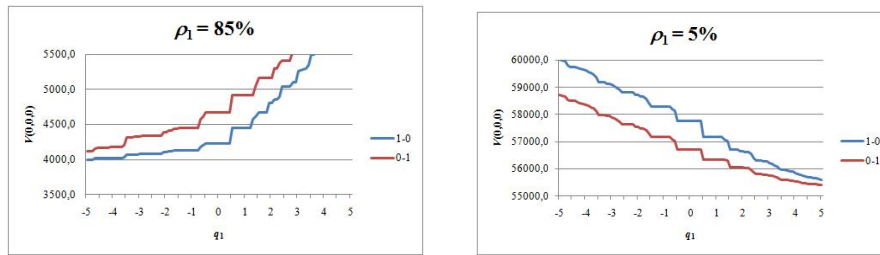


Fig. 3 Cost evolution.

Assume that for System 5 we have $\rho = 0.90$ and consider two cases for ρ_1 , 85% and 5%, which are the cases of maximum observed deviation between the optimal cost and the cost achieved under the $Gc\mu$ -rule. In Figure 3 we present the evolution of $V(0,0,0)$ as a function of q_1 for two pairs of values for q_2 and r_2 . Values were computed for steps of 0.1 for q_1 . The left plot displays the evolution for $\rho_1 = 85\%$ and on the right for $\rho_1 = 5\%$. The curve labeled "1-0" represents the case where $q_2 = 1$ and $r_2 = 0$, whereas the curve labeled "0-1" refers to $q_2 = 0$ and $r_2 = 1$. The first observation on these two plots concerns the dual nature of the behavior. In the left plot, for fixed q_1 , the lower costs are achieved for the "1-0" case, while on the right plot the lower costs are achieved for the "0-1" case. Also, cost is non decreasing with q_1 on the left and non increasing on the right.

The fact that the curves are non convex should not be a surprise given the discrete nature of the problem, since some minor change on q_1 may not produce any significant difference on the switching curve for integer values of the state space.

The behavior here displayed has been observed in all systems we tested. Taking the left plot as an instance, one should expect the cost to keep dropping as q_1 decreases down to some point, after which it should start increasing again. What we need to stress here is the fact that for high values of ρ_1 and fixing q_2 and r_2 there should be a value for q_1 where the curve reaches its minimum value, and the value of q_1 which achieves it is negative. On the right plot, there should also be a value of q_1 after which the cost should become non decreasing and that turning point is reached for positive values

of q_1 . This behavior is consistent with the shifts observed for the optimal switching curves, presented earlier.

Note also that any horizontal line crosses the two displayed cost curves or none of them. For fixed values of q_2 and r_2 , this means that any cost within the range of achievable costs can be reached with some choice of q_1 . This naturally includes the minimal value achievable under the policy produced by the $\Delta c\mu$ -rule.

In an effort to investigate how close to the optimal one could get with the $\Delta c\mu$ -rule, we conducted a series of line searches for different values of the parameters and came to a striking and elegant numeric coincidence. Maintaining the constraint that $q_1 + r_1 = 1$, we can restrict the search effort by imposing the following constraints.

$$\begin{cases} q_1 = r_2 \\ r_1 = q_2 \end{cases} \quad (21)$$

In Figure 4 we present the evolution of the percent deviation of $V(0, 0, 0)$ achieved relative to the optimal value as a function of q_1 when constraint (21) is enforced. We get very close to the optimal value on both situations. More specifically, for $\rho_1 = 85\%$ and with $q_1 = -2.5$, we achieve a cost of 3,920.74 while the optimal value is 3,920.73. When $\rho_1 = 5\%$ and $q_1 = 3.6$, we get a cost of 55,346.6 and the optimal value is 55,319.6. We limited the line search to steps of 0.1. Therefore, one can conjecture that it may be possible to get even closer. However, the displayed results are already sufficiently good to make our case.

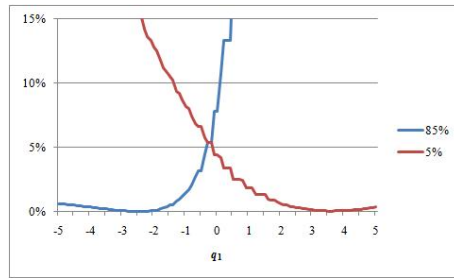


Fig. 4 Deviation to optimal value enforcing constraint (21).

Constraint (21) was unexpected when we initiated the study, but the fact that it holds is a strong mark of elegance. Given the fact that these parameters are functions of the individual loads, when we are dealing with only two classes of customers it makes a lot of intuitive sense that it should be this way. Therefore, the problem of identifying the optimal parameters for the $\Delta c\mu$ -rule reduces to a pure line search. Table 7 presents a sample of the best achieved performances for System 5 with varying ρ_1 . For each value of ρ_1 we display the value of q_1 which achieves the best performance and the percent deviation of the $Gc\mu$ and $\Delta c\mu$ -rules relative to the optimal value of $V(0, 0, 0)$. Although there is a range of loads for which the $Gc\mu$ -rule achieves highly acceptable performances, the table shows that it is always possible to do better by tuning the parameters of the $\Delta c\mu$ -rule.

Before we move on there are a couple of issues that deserve discussion. Firstly, given the fact that the $Gc\mu$ -rule has been proved to be asymptotically optimal in heavy

Table 7 Performance comparison.

ρ_1	q_1	$Gc\mu$	$\Delta c\mu$
85%	-2.5	19.19%	$<< 0.001\%$
75%	-1.4	17.95%	$<< 0.001\%$
65%	-0.9	7.68%	$<< 0.001\%$
55%	-0.6	2.99%	0.01%
45%	-0.1	1.00%	0.019%
35%	0.2	0.20%	$<< 0.001\%$
25%	0.8	0.03%	$<< 0.001\%$
15%	1.2	0.40%	0.0092%
5%	3.6	2.53%	0.049%

traffic and given the numeric evidence here presented, there is a need to interpret this inconsistency. In the context of a multiclass system, we define loosely the concepts of *biased* and *unbiased* heavy traffic. We term as unbiased heavy traffic for K classes a situation where $\rho_i \approx \rho/K$ for $i = 1, 2, \dots, K$, and define as biased heavy traffic a situation where one or more classes are such that $\rho_i \gg \rho/K$ and the remaining classes are such that $\rho_i \ll \rho/K$. What we have seen for the specific case of $K = 2$ is that the $Gc\mu$ -rule performs best in unbiased heavy traffic. The traditional heavy traffic analysis methodology does not account for this difference between biased and unbiased heavy traffic. In fact, in the absence of a specifically stated characterization on the nature of the heavy traffic, one can only assume that for $K \gg 1$ the results derived are specific for unbiased heavy traffic. While in general the results may be valid in the majority of the contexts irrespective of the heavy traffic characterization, we believe this particular example suggests that future heavy traffic analysis will have to include a validation that takes into account a possible variation of the derived policies when the traffic is biased.

Secondly, although we have shown that the optimal policy is sensitive to the individual loads and that using the $\Delta c\mu$ -rule is a way to get very close to the optimal performance for any load, there is no simple way to determine the adequate parameters to achieve those performances. Because we have been unable to identify the exact relation between q_1 and ρ_i , determining the optimal value for q_1 is more complex than determining the optimal policy alone. To do the line search described above implies running the value iteration algorithm for a choice of q_1 while the cost obtained for each keeps going down. On the other hand, there is no such problem with the $Gc\mu$ -rule. To overcome this drawback we propose to analyze a subset of choices for q_1 .

Table 8 presents the percent deviation of cost for the subset of interest. We code each of the entries to facilitate a reference to them in the discussion. Although the results presented refer to System 5, the qualitative behavior presented has been verified across all systems tested. What we see is that when ρ_1 is high, the best cost in the subset is achieved by the entry *HL*, while when ρ_1 is low the best cost is achieved with entry *LH*. If we assign to each character of the code the meaning *H* as high and *L* as low, then the results displayed have an easy and interesting interpretation. That is, when ρ_1 is high, the first cut solution that improves over the $Gc\mu$ -rule with no computational burden is High for class 1 and Low for class 2, which relates to their relative position in terms of individual loads. Conversely, if ρ_1 is low, then the solution is Low for class 1 and High for class 2.

Therefore, for the general case one would expect the definition of three regions for ρ_1 : high, intermediate, and low. If the individual load of class 1 falls into the high

Table 8 Deviation from optimal for a subset of policies.

q_1	r_1	q_2	r_2	Code	ρ_1	
					85%	5%
1	0	1	0	<i>LL</i>	13.35%	3.37%
1	0	0	1	<i>LH</i>	25.27%	1.86%
0	1	1	0	<i>HL</i>	7.83%	4.25%
0	1	0	1	<i>HH</i>	19.19%	2.53%

region, the choice should be *HL*; if it falls in the low region, the choice should be *LH*; and if it falls in the intermediate region, the choice could either be *LL* or the *Gcμ*-rule. When traffic is unbiased one can say that both classes have a low individual load, thus justifying the choice for the intermediate region. Determining the specific cutoff values to define the three regions can be done in a qualitative and loose manner. The reason for this is as follows. Even if range limits are slightly off, the resulting policy for the whole spectrum of values for ρ_1 is definitely better than just using the *Gcμ*-rule all the time.

Although we are not presenting any specific numeric evidence for preemptive systems, the results follow the same general structure just discussed. The only notable issue to remark here concerns the fact that the optimal switching curves differ for each system, depending if we allow preemption or not. Recall that for linear costs the switching curves are exactly the same.

5 Conclusions

In this paper, we have provided numeric evidence that the optimal policy for the single server scheduling problem, when costs are convex, depends on the individual load each class imposes on the server. We restricted our analysis to systems serving only two classes of customers. We formulated a set of related problems for which we were able to derive the optimal policy, and used the knowledge these problems provided to propose an alternative generalization of the *cμ*-rule. This new generalization, designated as the $\Delta c\mu$ -rule, relies on a composition of first order differences of the single stage cost function and is a function of the individual loads. We provided numeric evidence of near optimality for the $\Delta c\mu$ -rule. Given that tuning the parameters for this rule is more time consuming than finding the optimal policy, we proposed an approximation to it that can be obtained without any computational effort. This works by dividing the load space into three regions: High, Intermediate, and Low. Although we lose the near optimality, we still obtain better performances than the generalized *cμ*-rule of Mieghem, [16]. If traffic is biased the performance deviations tend to be higher in Mieghem's generalized rule. The performance deviations to the optimal are very small for unbiased traffic. Our $\Delta c\mu$ -rule can be fine tuned for any traffic condition, where it achieves near optimal performance in all cases tested.

Several different directions for future research can be foreseen. We focused on single server and two classes. So, a natural development would be to consider pools of servers and more classes of customers. We believe that the optimal policies should keep a similar structure to what was here presented, in terms of their dependence on the individual loads. However, the extension of the $\Delta c\mu$ -rule to more classes needs to be investigated, together with its potential performance gains.

References

1. P. S. Ansell and K. D. Glazebrook and J. Nio-Mora and M. O’Keeffe, Whittle’s index policy for a multi-class queueing system with convex holding costs, *Mathematical Methods of Operations Research*, Vol. 57, No. 1, 21-29 (2003)
2. J.S. Baras and A.J. Dorsey and A.M. Makowsky, Two competing queues with linear costs: the μc rule is often optimal, *Advances in Applied Probability*, Vol. 17, 186-209 (1985)
3. D.P. Bertsekas, *Dynamic Programming and Optimal Control*, Athena Scientific, (1995)
4. C. Buyukkoc and P. Varaiya and J. Walrand, The $c\mu$ rule revisited, *Advances in Applied Probability*, Vol. 17, No. 1, 237-238, (1985)
5. C.G. Cassandras, *Discrete Event Systems: Modeling and Performance Analysis*, Richard D. Irwin, Inc. and Aksen Associates, Inc. (1993)
6. D.R. Cox and W.L. Smith, *Queues*, Chapman & Hall (1961)
7. J. G. Dai and W. Lin, Maximum pressure policies in stochastic processing networks, *Operations Research*, Vol. 53, No. 2, 197-218 (2005)
8. J. G. Dai and W. Lin, Asymptotic optimality of maximum pressure policies in stochastic processing networks, *Annals of Applied Probability*, Vol. 18, No. 6, 2239-2299 (2008)
9. J.M Harrison, Dynamic scheduling of a multiclass queue: discount optimality, *IEEE Transactions on Automatic Control*, Vol. 23, No. 2, 270-282 (1975)
10. J.M. Harrison and A. Zeevi, Dynamic scheduling of a multiclass queue in the Halfin-Whitt heavy traffic regime, *Operations Research*, Vol. 52, No. 2, 243-257 (2004)
11. A. Mandelbaum and A.L. Stolyar, Scheduling flexible servers with convex delay costs: heavy-traffic optimality of the generalized $c\mu$ -rule, *Operations Research*, Vol. 52, No. 6, 836-855 (2004)
12. M. Pinedo, Stochastic scheduling with release dates and due dates, *Operations Research*, Vol. 31, 559-572 (1983)
13. M. Rebello and Z. Fernandes and C. Bispo, Simulating activity networks in JAVA, *Proceedings of the 6th EUROSIM Congress on Modelling and Simulation*, Ljubljana, Slovenia (2007)
14. R. Righter, *Scheduling. In Stochastic Orders* (M. Shaked and J.G. Shanthikumar eds.), Academic Press (1994)
15. W.E. Smith, Various optimizers for single-stage production, *Naval Research Logistics Quarterly*, Vol. 3, 59-66 (1956)
16. J.A. van Mieghem, Dynamic scheduling with convex delay costs: the generalized $c\mu$ rule, *The Annals of Applied Probability*, Vol. 5, No. 3, 808-833 (1995)

Low Energy Scheduling with Power Heterogeneous Multiprocessor Systems

Richard Dobson · Kathleen Steinhöfel

Abstract *In this paper we consider low energy scheduling for power heterogeneous multiprocessor systems. This is a fast developing area that is of great importance and is currently being researched by both industry and academia. This is partially because heterogeneous multiprocessor systems are more prevalent than one would expect, homogeneous systems often become heterogeneous due to manufacturing inaccuracies. We have developed an algorithm which transforms any multiprocessor system into a Virtual Single Processor (VSP); the VSP is a result of combining the component processors in an pareto-optimal way with regards to overall system speed and energy consumption. Using the VSP approach existing speed scaling algorithms (which were designed for single processor systems) can be applied to heterogeneous multiprocessor systems. In this study we focus on algorithms which which minimise $\sum \text{Flow} + \text{Energy}$ in conjunction with our VSP approach as $\sum \text{Flow} + \text{Energy}$ minimisation is applicable to many real world situations.*

Keywords Virtual Single Processor, VSP, Dynamic Speed Scaling, Energy, Flow, Power Heterogeneous Multiprocessor Systems, Scheduling

1 Introduction

Energy consumption is a important consideration when designing computer systems; especially when for use as a mobile device such as a smart phone, which needs to provide both high performance and good battery life. One of the largest drains of energy in a computer system is the processors. In most modern processors energy consumption and processing speed are intrinsically linked, this is normally through the relationship $\text{Power} = \text{Speed}^a$ where a is a constant which is typically between 2 and 3. A very effective way of reducing the amount of energy a processor uses is by lowering it's operational speed, for this we can use Dynamic Speed Scaling (DSS).

Richard Dobson
King's College London, Department of Informatics
E-mail: richard.dobson@kcl.ac.uk

Kathleen Steinhöfel
King's College London, Department of Informatics
E-mail: kathleen.steinhofel@kcl.ac.uk

DSS allows the operating frequency of a processor to be modified easily during use; due to the polynomial relationship between speed and power a small reduction in processor speed can result in a large reduction in energy consumption. This method allows single and multiple processor computer systems save large amounts of energy and is an essential tool for any developer wishing to lower the energy consumption of a computer system. DSS allows us to lower the operational speed of the processors but we need to combine this with a objective function which controls how much the processors should be slowed down by. This is by no means a trivial task as we need to ensure that energy is not needlessly wasted processing jobs too quickly and the operation of the computer system is not compromised by processing jobs too slowly.

1.1 Related Work

Yao, Demers and Shenker [12] presented the first method which uses DSS, they suggested using hard job deadlines and propose a natural algorithm which sets the speed of the processor to the minimum possible such that all deadlines are met. In their paper they suggest a number of approximation algorithms which aim to implement the natural algorithm. This paper initiated research into the area of deadline scheduling with many algorithms and approaches being developed (see [1] for a recent review). Deadline scheduling is a good solution but in real world situations we find that many jobs do not have natural deadlines and artificially imposing deadlines can lead to an increase in energy consumption as jobs are completed more quickly than needed.

Some years later Albers and Fujiwara [2] presented an alternative solution which does not rely on deadlines but attempts to balance the quality of service against power consumption. The authors attempt to minimise $\sum Flow + Energy$, this balances the two conflicting values of $Energy$ against $\sum Flow$ where the $Flow$ of each job is the time between it's release and completion. At the time of publication this was revolutionary as it means that each jobs no longer need to be assigned (potentially artificial) deadlines. Albers and Fujiwara developed a batch processing algorithm which was proved in [5] to be 4-Competitive compared to optimal F+E minimisation. There have subsequently been many algorithms which have built on the concept of $\sum Flow + Energy$ minimisation.

Bansal, Pruhs and Stein [5] first extended the concept to include job weights $\sum WeightedFlow + Energy$ (which we will now refer to as FW+E for convenience); this allows for some jobs to be marked as more important which means they will be processed more quickly. One of the most recent results presented by Bansal, Pruhs and Chan [4] allows the algorithm they develop to be applied to any processor with an arbitrary power function providing it is strictly non decreasing and still adhere to a bounded competitive ratio. The algorithm states that the speed of the processor should be determined by amount of jobs which have not finished processing, as in formula 1. This algorithm has been proved to be $(3 + \epsilon)$ -competitive for any non decreasing power function.

$$s_a^t = \begin{cases} P^{-1}(n_a^t + 1) & \text{if } n_a^t \geq 1; \\ 0 & \text{if } n_a^t = 0. \end{cases} \quad (1)$$

Andrew, Wierman and Tang [3] recently presented an alternative algorithm for the same problem as Bansal et. al. but with an improved competitive ratio. The algorithm considers a wider range of power function and obtains a competitive ratio of $(2 + \epsilon)$;

the authors also show that there exists some trade off functions no algorithm can be better than (2)Competitive.

Bunde [7] extended the existing knowledge of minimising $F+E$ for the single processor to the uniform multiprocessor environment. In the paper Bunde describes an offline algorithm which is optimal for the minimisation of $F+E$.

Lam et al extended existing knowledge of single processor $F+E$ algorithms to the homogeneous multiprocessor situation in their paper [11]. The authors looked at the online problem and suggested an algorithm Classified Round Robin (CRR) where jobs are distributed fairly evenly between all processors using a weighted round robin; individual processors are then left to manage their own speed using a single processor $F+E$ minimisation algorithm. Less than a year later Lam et al released [10] which presents a slightly improved version of the algorithm.

1.2 Heterogeneous Multiprocessor Systems

Homogeneous Multiprocessor systems, where all processors are equal, have become increasingly common through multi-core processors and have the ability to be incredibly efficient, especially if all jobs are fully paralysable but this rarely is the case. In many situations particular threads are strictly sequential and are required to be processed on just one processor, this can be a problem for homogeneous multiprocessor systems as long chains of sequential jobs can cause a block in the whole system delaying jobs significantly. One potential solution to this problem is using a combination of a set of low performance but energy efficient processors to process paralysable jobs and a small number of high performance but less efficient processors which we can use to process threads which are sequential or high priority jobs. If we were to use this solution the multiprocessor configuration would no longer be homogeneous but heterogeneous instead.

Heterogeneous Multiprocessor Systems are currently not common in consumer hardware but this is likely to change as our understanding of how to best exploit this type of infrastructure improves. If structured correctly heterogeneous multiprocessor systems have the potential to be very adaptable allowing the computer to be both energy efficient and computationally powerful. If a homogeneous multiprocessor system encounters a large amount of jobs which must be ran sequentially then one processor must run very fast to maintain a good quality of service. It is often the case in a multi-core processor (the most common example of a homogeneous multiprocessor system) that all cores must operate at the same speed, this clearly would increase the energy consumption by a factor of n where n is the number of cores. This is the advantage of running a heterogeneous multiprocessor system as we can have a variety of different processor which allow us to deal with both jobs which can be run in parallel and sequential jobs in an efficient manor.

The use of heterogeneous processor systems is not a new idea as they have been used in processing clouds and server systems for many years but the new innovation is using heterogeneous processor systems in an energy efficient manor. In 2008 Bower, Sorin and Cox wrote a paper [6] which outlined the importance of considering the energy efficient heterogeneous multiprocessor situation. The authors highlight that even in a homogeneous multiprocessor system using DSS is often a heterogeneous at run time. They identify 3 main hurdles for heterogeneous multiprocessor scheduling:

1. The OS must find the status of the processor,

2. The OS must find the demands of each job and
3. The OS must match jobs to processors as well as possible using the available information.

Gupta, Krishnaswamy and Pruhs suggested the first solution to this problem in [9], the authors tackle the 3rd problem identified in [6] of organising which jobs should be processed on which processor and when. They suggest a simple algorithm which they split into three parts for simplicity:

1. Job Selection (which job should be ran on each processor): Highest Density First
2. Speed Scaling (what speed should each processor run at): The speed is set so the power is the fractional weight of the unfinished jobs
3. Assignment (which processor should each job be assigned to): A new job is assigned to the processor that results in the least increase in the projected future weighted flow, assuming the adopted speed scaling and job selection policies, and ignoring the possibility of jobs arriving in the future

The approach which Gupta et al. suggest involves distributing jobs to various processors based on which one would provide the smallest increase in the projected flow providing no more jobs arrive and the speed of the processors does not change. The processors are then allowed to manage their own speed based on the volume of work they are carrying and the speed scaling policy defined by the algorithm. In the paper the authors prove this algorithm to be "scalable for scheduling jobs on a heterogeneous multiprocessor with arbitrary power functions to minimize the objective function of weighted flow plus energy" [9].

2 Low Energy Scheduling for Heterogeneous Multiprocessor Systems

In the section above we discuss existing research which looks at multiprocessor scheduling specifically concerning power heterogeneous multiprocessor systems. In [9] the authors present a solution for power heterogeneous multiprocessor systems using weighted Flow + Energy as their objective function. This solution has been shown to be theoretically sound for the model which they consider. If we consider a real life multiprocessor computer system we often find that there are constraints which are applied to the processors. For example many multi-core processors require the cores to always run at the same speed. The Gupta et. al approach is not compatible with this situation as the processor speed is linked to the number of jobs.

The Gupta et al algorithm also requires a large amount of runtime computation. Each time a job needs to be assigned to a processor it must find out which processor will provide the smallest increase in the weighted flow. This requires a small amount of processing for just one job but over time this could potentially add up to a large amount of computation.

We present a solution that has the ability to overcome both of these issues. Our solution suggests that we should form a 'Virtual Single Processor' (VSP). The VSP is essentially a collection of processors which have been combined together in an efficient way to form what appears to be a single processor. We then present the VSP (as a single processor) to a DSS algorithm which controls the speed of the overall VSP and specifies which job should be processed first. The VSP in turn translates the overall VSP speed into speeds for each processor which when combined equal the VSP speed.

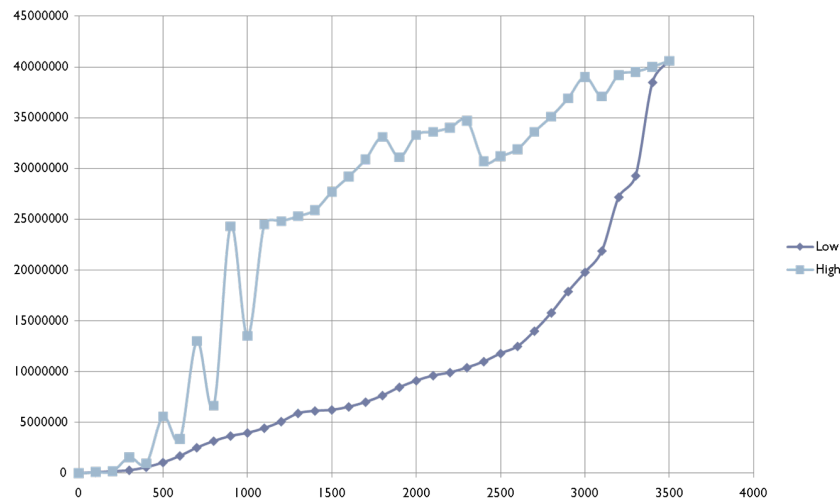


Fig. 1 A graph showing the relationship between system speed and power consumption for the best case and worst case processor combinations.

3 The Virtual Single Processor

When we are dealing with a multiprocessor system we can consider it in one of two ways. We can either distribute the jobs at system level and control the processor speeds at individual processor level (as in [10] and [9]), or as we suggest in this paper we can consider controlling processor speed at system level and allow the processors to request a new job when they have completed their current job.

We consider a small example of a 4 processor system (P0, P1, P2, P3) where each processor has a set of speeds and a simple power function in the form of $P = S^a$, the attributes of which are outlined below.

P0 (0, 200,300,400, 500) $a=2.3$

P1 (0, 600, 700,800,900) $a=2.35$

P2 (0, 100, 300, 500, 700, 900) $a=2.5$

P3 (0, 1200) $a=2.2$

In this simple example there are many ways to combine these processors with 300 unique combinations and 14 different combinations which make up the overall system speed of 1400 alone. If we consider the best and worst ways of achieving the system speed of 1400 (with regards to energy) we find that the worst case uses 487% of the energy consumed by the most efficient combination. If we look at Figure 1 we can see the difference between the most and least energy is largest in mid range speeds with the graph converging at either end of the system speed range. All processors must be at speed 0 for the system speed to be 0 and all at maximum speed for the system speed to be maximised. This simple example highlights how crucial it is to make the correct processor speed selections. If we pre-compute the optimal combinations before attempting to use a multiprocessor system we can reduce the chance of using a less efficient processor combination.

To find out which processor combinations are best we first define rules which specify which allow us to identify which combinations we wish to discard and which to keep.

We first specify that a combination can only be part of the optimal VSP if there is no other combination which provides a higher system speed for the same amount of energy or the same system speed for less energy. This is encapsulated in Observation 1, the proof of which is omitted due to the intuitive correctness as the optimal VSP should minimise the energy consumption for any system speed.

Observation 1 *An optimal VSP has a strictly non decreasing power function where no feasible processor combination exists which consumes less energy for any system speed.*

Observation 1 severely reduces the amount of combinations but this does not always result in the set of optimal combinations. The final method which we use to further reduce the lower envelope of the power function is speed simulation which was first described in [4]. Bansal first defined this method in order to show that a processor with a restricted set of speeds could simulate any speed in the range of 0 - max by alternating between 2 different speeds. For example if we have a processor with 2 available speeds (0 and 10) then we can simulate the speed 5 by alternating between the two available speeds in equal amounts. If we wished to simulate the speed 7.5 then we would use speed 10 for $\frac{3}{4}$ of the time and speed 0 for $\frac{1}{4}$ of the overall time. We use this method in order to potentially lower the energy consumption of the VSP.

Observation 2 *If a VSP speed s can be simulated by alternating between two different speeds and the simulated speed consumes less energy then s is not part of the optimal VSP.*

Once both Observation 1 and 2 have been applied to a set of processor combinations the combinations left form the optimal VSP. Figure 2 shows an instance in which Observation 2 is used to lower the overall power function of the resulting VSP. After applying both Observation 1 and Observation 2 we have computed the optimal VSP as it is not possible achieve a higher system speed for any energy level.

The VSP provides a level of abstraction between the single processor algorithm and the multiprocessor system. This abstraction allows us to hide the complexity of the multiprocessor system behind the VSP front. We can hide complex requirements such as processors which always need to operate at the same speed by first producing a small VSP and then feeding this into the overall VSP as if it were a single processor.

By calculating the VSP we can also remove the burden of calculating which processor is best for each job. This is made possible as the VSP hides the fact that more processors exist and only assigns jobs when a processor has a speed greater than 0 and no job. This means that there is no need for the algorithm to perform costly calculations at run time.

4 Our VSP Algorithm

In this section we define our dynamic programming algorithm DynaVSP which calculates an optimal VSP given a set of processors as an input.

Based on a simple implementation we find the algorithm to have a worst case running time of $O(m^2 \log n)$ based on n processors and with each processor having m speeds.

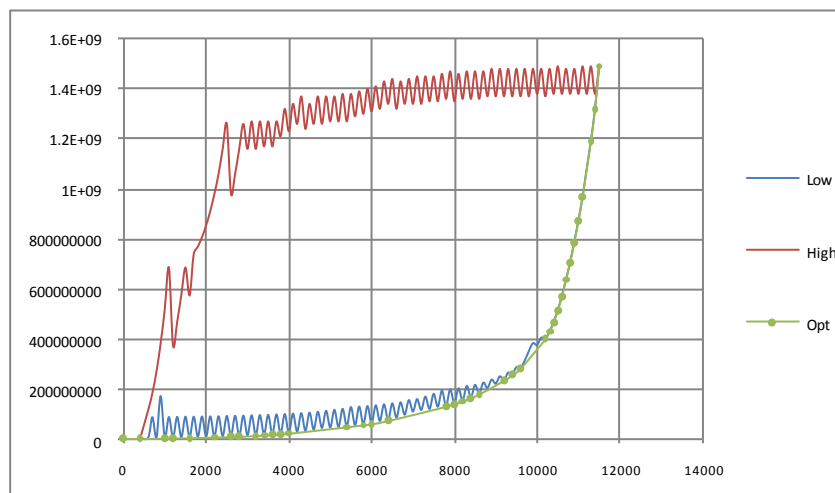


Fig. 2 A graph showing the relationship between system speed and power consumption including energy spikes and optimal system speeds marked by dots

DynaVSP

Input: A list of processors P of length n

```

if  $n < 2$  then
    return processor in  $P$ 
end if
if  $n == 2$  then
    Construct a VSP with all combinations of  $P[0]$  and  $P[1]$ 
    Remove inefficient combinations according to Observation 1 and 2
    Return VSP
end if
if  $n > 2$  then
    Return DynaVSP( [DynaVSP( $P[0..n/2]$ ), (DynaVSP( $P[(n/2) + 1..n]$ ))] )
end if

```

Output: A single VSP

5 Using the Virtual Single Processor

Once we have constructed our VSP we need three things, the job selection policy and the speed scaling policy of the single processor algorithm and whether the computer system will allow migration between processors or not. The third point is crucial to job selection and processor speed changes so we will discuss these two options separately.

5.1 Migratory

Incoming jobs are sorted according to their ranking as judged by the job selection policy, the job with the highest ranking from all jobs which have not yet finished processing is always assigned to the fastest processor, the second highest ranking with the second fastest processor and so on. We ensure that this is always the case even when more

jobs arrive or processor speeds change. This makes sure that the job with the highest priority always finishes quickly. The speed scaling policy is used in conjunction with the system power function to determine what speed our system should operate at, this is then translated into individual processor speeds by the VSP. If a processor is directed to use speed 0 then the job it is currently processing is suspended and returned to the list of incoming jobs.

5.2 Non Migratory

Once again the incoming jobs are sorted according to their ranking as judged by the job selection policy but we also keep a note of 2 things for each processor, how much time is required to finish the current job being processed if the processor speed stays constant $P_i T$ and what the current speed of the processor is $P_i S_c$. We then calculate which processor will allow the highest priority job to finish first providing processor speeds stay constant and the second highest to finish second and so on. Jobs are then assigned to the 'correct' processors when they become available. When the speed scaling algorithm decides that the system speed should change the VSP converts this into individual processor speed changes; if the speed of a processor should rise then this happens straight away, if the speed of a processor should drop this action is taken after the processor has finished processing the current job. This ensures that no job is trapped on a processor which has a speed 0 as this could result in the job never being finished.

6 VSP System Analysis

It is important that we do not see the VSP as a complete scheduling algorithm. It is a platform which allows scheduling algorithms to be applied to or developed for heterogeneous multiprocessor system more easily. In this section we combine the VSP system with the speed scaling policy and job selection policy from the Gupta algorithm. We then compare the two approaches using simulations. If the results from the two differing approaches are similar then this will show that the VSP has the potential to be a favorable alternative.

We define 2 multiprocessor systems S_1 and S_2 which have identical processor configurations and neither allow migration. S_1 is used in conjunction with the Gupta et al. approach and speed scaling algorithm A as defined in [9]; S_2 is used as a VSP and also uses speed scaling algorithm A. The VSP approach takes S_2 and uses the VSP algorithm to convert it into an optimal VSP which we will call V_2 .

If we consider a batch of t tasks arriving at over time we can compare how each approach will deal with these. The Gupta approach will sort the jobs by their density and then calculate which processor will provide the least increase in projected flow for each job, the job is assigned to this processor. Each processor will calculate what speed it should be running at based on the fractional weighted flow of it's work. The VSP approach will sort the jobs by their density and then calculate what the speed of the VSP should be, the VSP will then instruct the processors what speed they should be running at. Jobs are assigned to a processor if it's speed is greater than 0 and it does not already have a job. Jobs with higher priorities will be assigned to faster processors.

The VSP approach has allowed us to remove the majority of the computation from run time, this is possible because we precompute the processor configurations. Once we have the processor configurations all we have to do at run time is look up the desired system speed based on the quantity of work and implement the indicated processor speeds. Job distribution is then simplified by putting the highest density job on the fastest processor. This indicates that in real life situations the VSP approach could save a considerable amount of effort at run time over the Gupta et al approach.

To test the overall performance of the VSP system we developed simulations of both approaches and ran a number of tests with a variety of processor configurations and job sets. In this section we will highlight the tests regarding a processor configuration outlined in [8]. The suggested processor configuration has x high powered processors, $2x$ medium powered processors and $4x$ low powered processors.

We use the processor configuration from [8] where $x = 1$. $1 \times$ High speed processor: $s = \{0, 1000, 2000\}$ $a = 2.8$, $2 \times$ medium speed processors: $s = \{0, 250, 500, 750, 1000\}$ $a = 2.55$, $4 \times$ low speed processor: $s = \{0, 50, 100, 150, 200, 250\}$ $a = 2.25$. We split the test data into 3 different categories all of which contain jobs with random weights and sizes:

- Immediate; all jobs are all released at time 0
- Uniformly random; jobs are released randomly over time up to time x
- Peaks and troughs; jobs are released in surges which are similar to the action of a computer system up to time x

Covering these categories gives us a good indication of overall system performance.

After running our simulations we found some very interesting results. We report the values below in terms of VSP performance in comparison to the Gupta et. al. algorithm. Upon extracting the data from the simulations we found that there were some jobs which were severely throwing the average finish time for both the VSP and Gupta algorithms. The few jobs which have been removed have both a very low weight and a very large size. After removing these we recalculate the averages and have reported this in Figure 3.

The results prove to be very interesting. They show us that the VSP platform is very promising, this has allowed us to combine the Gupta et. al. algorithm with the VSP platform and achieve very competitive results. Although the energy consumption is larger, the $\sum \text{Flow} + \text{Energy}$ is lower. This is the aim of the algorithm, to minimise the $\sum \text{Flow} + \text{Energy}$ so this has been very successful.

7 Conclusion

In this paper we present the Virtual Single Processor approach to low energy scheduling for power heterogeneous multiprocessor computer systems. We show that the VSP approach is theoretically sound and can be used with existing technologies soliciting strong results. In future papers we hope to expand this area by looking at variations of the VSP system that are split into sub VSPs some that are designed to handle large / sequential jobs and others for smaller parallel jobs.

References

1. Susanne Albers. Algorithms for Dynamic Speed Scaling. In Thomas Schwentick and Christoph Dürr, editors, *28th International Symposium on Theoretical Aspects of Com-*

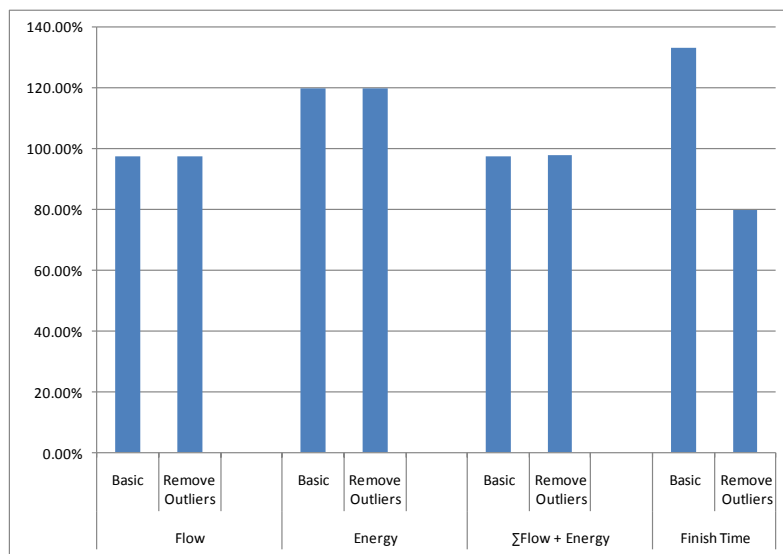


Fig. 3 A graph showing the average results from the simulations and the average results with outliers removed

- puter Science (STACS 2011), volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–11, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Trans. Algorithms*, 3, November 2007.
 - Lachlan L.H. Andrew, Adam Wierman, and Ao Tang. Optimal speed scaling under arbitrary power functions. *SIGMETRICS Perform. Eval. Rev.*, 37:39–41, October 2009.
 - Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 693–701, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.
 - Nikhil Bansal, Kirk Pruhs, and Cliff Stein. Speed scaling for weighted flow time. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 805–813, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
 - Fred A. Bower, Daniel J. Sorin, and Landon P. Cox. The impact of dynamically heterogeneous multicore processors on thread scheduling. *IEEE Micro*, 28:17–25, May 2008.
 - David P. Bunde. Power-aware scheduling for makespan and flow. *J. of Scheduling*, 12:489–500, October 2009.
 - Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Nonclairvoyantly scheduling power-heterogeneous processors. In *Proceedings of the International Conference on Green Computing*, GREENCOMP '10, pages 165–173, Washington, DC, USA, 2010. IEEE Computer Society.
 - Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Scalably scheduling power-heterogeneous processors. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming*, ICALP'10, pages 312–323, Berlin, Heidelberg, 2010. Springer-Verlag.
 - Tak-Wah Lam, Lap-Kei Lee, Isaac To, and Prudence Wong. Improved multi-processor scheduling for flow time and energy. *Journal of Scheduling*, pages 1–12, 2009. 10.1007/s10951-009-0145-5.
 - Tak-Wah Lam, Lap-Kei Lee, Isaac K. K. To, and Prudence W. H. Wong. Competitive non-migratory scheduling for flow time and energy. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, SPAA '08, pages 256–264, New York, NY, USA, 2008. ACM.

12. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, FOCS '95, pages 374–, Washington, DC, USA, 1995. IEEE Computer Society.

A Framework for Implementation-Oriented Staff Scheduling

Eyjólfur Ingi Ásgeirsson • Jari Kyngäs • Kimmo Nurmi • Martin Stølevik

Abstract This paper presents a general framework for an implementation-oriented, highly constrained staff scheduling problem that is modeled from the requirements of various lines of business and industry. The presented model builds up a solid foundation for wide variety of real-world staff scheduling scenarios. We hope the presented modeling issues assist academics in getting their research results implemented into commercial systems. We define a staff scheduling problem, introduce the necessary terminology, discuss research guidelines and detail the constraints of the problem. A set of artificial and real-world instances derived from the actual problems solved for various companies are presented. We publish the best solutions we have found, and invite the staff scheduling community to challenge our results. We believe that the instances will help researchers to test the implementation value of their solution methods. The instances are available online.

1 Introduction

Staff scheduling is a difficult and time-consuming problem that every company or institution that has employees working on shifts or on irregular working days must solve. The staff scheduling problem has a fairly broad definition. Most of the studies focus on assigning employees to shifts, determining working days and rest days or constructing flexible shifts and their starting times. Different variations of the problem are NP-hard and NP-complete [1]-[7] and thus extremely hard to solve. The first mathematical formulation of the problem based on a generalized set covering model was proposed by Dantzig [8]. Good overviews of staff scheduling can be found in [9]-[11].

Nurse rostering [12] is by far the most studied application area in staff scheduling. Other successful application areas include airline crews [13], call centers [14], check-in counters [15], ground crews [16], nursing homes, call centers and airport ground services [17], postal services [18] and transport companies [19]. Recent successful algorithms for staff scheduling include ant colony optimization [20], dynamic programming [21], constraint programming [22], genetic algorithms [23], scatter search [24], hyperheuristics [25], integer programming

Eyjólfur Ingi Ásgeirsson
Reykjavik University, School of Science and Engineering, Iceland
E-mail: eyjo@hr.is

Jari Kyngäs
Satakunta University of Applied Sciences, Finland
E-mail: jari.kyngas@samk.fi

Kimmo Nurmi
Satakunta University of Applied Sciences, Finland
E-mail: cimmo.nurmi@samk.fi

Martin Stølevik
SINTEF ICT, Department of Applied Mathematics, Norway
E-mail: martin.stolevik@sintef.no

[26], metaheuristics [27], simulated annealing [28], tabu search [29] and variable neighborhood search [30].

There are basically four reasons for the current interest in staff scheduling. First, public institutions and private companies around the world have become more aware of the possibilities for decision support technologies, and they no longer want to handle the schedules manually. Second, human resources are one of the most critical and most expensive resources for these organizations. Careful planning can lead to significant improvements in productivity. Third, good schedules are very important for the welfare of the staff. Besides increasing employee satisfaction, effective labor scheduling can also improve customer satisfaction. Finally, new algorithms have been developed to tackle previously intractable problem instances, and, at the same time, computer power has increased to such a level that researchers are able to solve large real-world problems. One further significant benefit of automating the scheduling process is the considerable amount of time saved by the administrative staff involved.

The focus of this paper is to introduce a general framework for an implementation-oriented, highly constrained staff scheduling problem that is modeled from the requirements of various lines of business and industry. The presented model builds up a solid foundation for wide variety of real-world staff scheduling scenarios. In Section 2 we introduce the necessary staff scheduling terminology. Section 3 discusses research guidelines and presents a model for an implementation-oriented staff scheduling problem. The next two sections show that both existing problems and artificial ones can be expressed by the model. In Section 4 we present a set of artificial test instances and in Section 5 we present real-world instances derived from the actual problems solved for various lines of business. We publish the best solutions we have found and invite the staff scheduling community to challenge our results.

2 Staff Scheduling Terminology

Staff scheduling consists of assigning *employees* to tasks and shifts over a period of time according to a given timetable. The *planning horizon* is the time interval over which the employees have to be scheduled. Each employee has a *total working time* that he/she has to work during the planning horizon. Furthermore, each employee has *competences* (qualifications and skills) that enable him/her to carry out certain *tasks*. Days are divided into *working days* (days-on) and rest days (*days-off*). A sequence of working days with one shift each day is called a *work stretch*. Each day is divided into *periods* or *timeslots*. A timeslot is the smallest unit of time. The length of a timeslot determines the granularity of the schedule. A *shift* is a contiguous set of working hours and is defined by a day and a starting period on that day along with a *shift length* (the number of occupied periods). Shifts are usually grouped in *shift types* - for example morning day and night shifts. Each shift may be composed of a number of tasks. A shift or a task may require the employee assigned to it to possess one or more competences. A *combined shift* is a collection of non-overlapping shifts; an employee assigned to a combined shift is assigned to all the shifts in the combined shift. If the combined shift is composed of non-contiguous shifts it is called a *split-shift*. A specific sequence of shifts for an employee is called a *stint*. A work schedule for an employee over the planning horizon is called a *roster*. A roster is a combination of shifts and days-off assignments that covers a fixed period of time. Figure 1 clarifies the terminology.

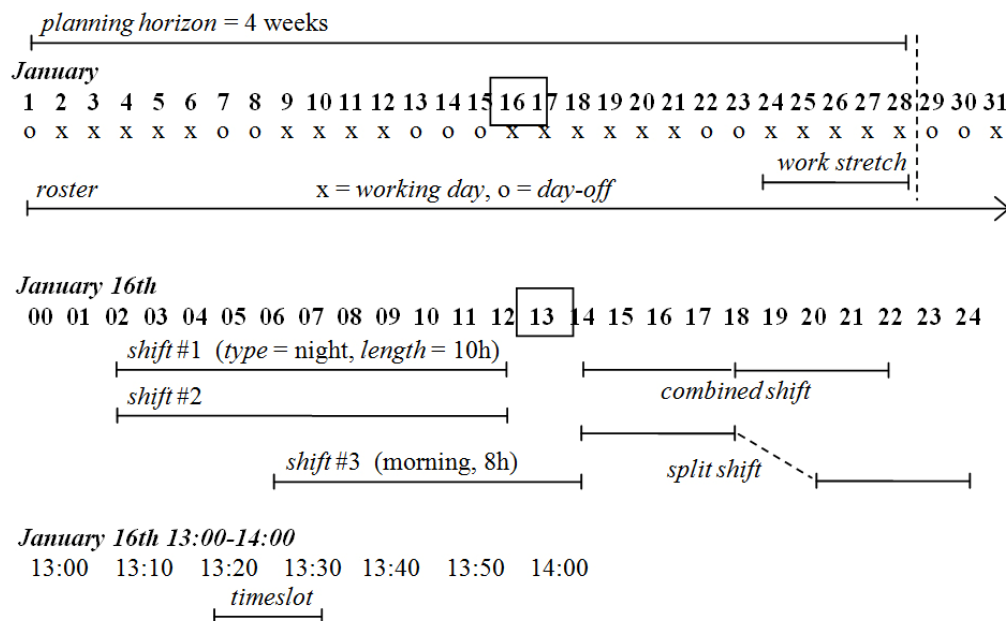


Figure 1: Staff scheduling terminology.

Cyclic schedules are such that all employees have the same basic schedule but start with a different day. In cyclic scheduling the goal is to find a schedule that is optimal for all employees. *Non-cyclic* schedules are individual schedules. In non-cyclic scheduling the goal is to find rosters that fulfill as many requests as possible. *Continuous* schedules arise in organizations that operate 24 hours a day and seven days a week - otherwise a schedule is called *discontinuous*. *Wrap-around scheduling* means that after the end of the planning horizon, the employees are expected to work the same roster all over again. If wrap-around scheduling is in use, some constraints must be checked outside the planning horizon by extrapolating the employees' rosters. If the process must take into consideration the shifts worked right before the beginning of the planning horizon, we call this *initial condition scheduling* and assume that the initial conditions are input to the problem. For example, decomposing a long time horizon into shorter time horizons which are solved sequentially can be a successful method in practical problems and would create initial conditions that must be handled.

	Mon		Tue		Wed		Thu		Fri		Sat		Sun	
	Early	Late	Early	Late	Early	Late	Early	Late	Early	Late	Early	Late	Early	Late
Aaron	1		1		1		1		1				1	
Betty	3		3		3		3				3		3	
Cecil		3		1			2		2		1		2	
David	2		2		1				1		2			2
Ellie			3		3		3		3		3		3	
Fred		2			2		2		3		2			1
Gary		1		2		2		1		2		1		

Table 1: An example and solution of staff scheduling problem.

Table 1 shows a solution for a simple one-week staff scheduling problem with seven employees, two shifts (early and late) in a working day and one of three tasks to be completed within a shift. Moreover, task 1 and 2 cannot be carried out by Betty and Ellie, a late shift

cannot be followed by an early shift on the next day, and each employee should have one day-off.

We classify the real-world staff scheduling process as given in Figure 2. *Demand modeling* is the process of determining the staffing levels, that is, how many employees are needed at different times over some planning horizon. In this presentation, demand modeling also includes determination of planning horizons, competence structures, regulatory requirements and other constraints. Demand modeling requires labor forecasting as well as hiring and budgeting decisions. *Shift design* is the process of determining the shift structure, tasks to be carried out in particular shifts and the competence needed in different shifts. The output of the demand modeling and the shift design is a mathematical model of the staff scheduling problem at hand.

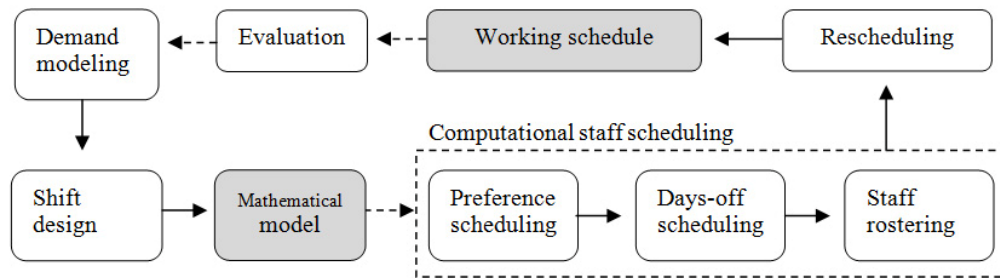


Figure 2: The real-world staff scheduling process.

In *preference scheduling*, each employee gives a list of preferences and attempts are made to fulfill them as well as possible. *Pure self-scheduling* refers to preference scheduling in which the employees are entirely responsible for the schedule generation. *Days-off scheduling* deals with the assignment of rest days between working days over a given planning horizon. *Staff rostering*, also referred as shift scheduling, deals with the assignment of employees to shifts. It can also specify the starting time and duration of shifts for a given day even though in most cases they are preassigned in shift design. In other words, days-off scheduling deals with working days and staff rostering deals with the working times of day. When days-off and shifts are scheduled simultaneously, the process is sometimes called *tour scheduling*. The name comes from the fact that we need to specify the hours of the day and days of the week through which each employee must travel. In this paper, the process of scheduling preferences, days-off and shifts is called *computational staff scheduling*. Computational staff scheduling is a key to increased productivity, quality of services, customer satisfaction and employee satisfaction. Other advantages include reduced planning time, reduced payroll expenses and ensured regulatory compliance.

Rescheduling deals with ad hoc changes that are necessary due to sick leaves or other no-shows. The changes are usually carried out manually. Finally, participation in *evaluation* ranges from the individual employee, through personnel managers, to executives. A reporting tool should provide performance measures in such a way that the personnel managers can easily evaluate both the realized staffing levels and the staff satisfaction. When necessary, the demand modeling and/or shift design can be reprocessed and focused, and the whole staff scheduling process restarted.

In the demand modeling we have to determine which phases in the computational staff scheduling are scheduled simultaneously. For example, scheduling days-off every tenth week and shifts every second week, enables the staff to plan their free time more conveniently. However, scheduling both at the same time every second week increases the probability of meeting the staffing levels without the need to hire part-time employees. The staff scheduling problems presented in this paper include at least two of the components of the computational staff scheduling. Some of them can be scheduled simultaneously. Alternatively, some of them can be solved first, and then the solution is given as a fixed input to the next component.

3 A Model For Implementation-Oriented Staff Scheduling Problems

There are hundreds of staff scheduling solutions commercially available and in widespread use. In that sense, the implementation-oriented staff scheduling approach is already standard practice in industry. However, we believe that there is still some gap between academic and commercial solutions. The commercial products may not include the best academic solutions. We define the implementation-oriented staff scheduling research as a research which raises

- 1) such modeling issues that have probably precluded academics from getting their research results implemented to commercial advantage, and
- 2) the collaboration between an academic researcher, a problem owner and an industry software vendor.

We have studied a significant number of theoretical and practical staff scheduling papers to realize the most important elements in real-world systems. This, combined with our experience in implementing staff scheduling software, forms the basis to the model. We believe that a considerable number of real-world staff scheduling scenarios can be modeled using the constraints presented in this section.

Most of the staff scheduling cases in which academic researchers have announced that they have closed a contract with a customer concern nurse rostering, see e.g. [27] and [31]-[37]. Hospitals tend to be very open about their operational details, enabling easy cooperation with academics who wish to publish the results of their work. We have experienced that nurse rostering cooperation between a commercial software vendor and academics work. The implemented plug-ins are a great sales pitch for the software vendors when they talk to potential customers.

The interest of the academic staff scheduling community has somewhat shifted to other application areas. Our goal is to pace this shift. Staff scheduling in various lines of business and industry is clearly somewhat different than in public hospitals. The overall interest to real-world staff scheduling has also increased: the number of presented staff scheduling papers has increased in two recognized scheduling conferences, PATAT and MISTA.

According to our experience, the best action plan for real-world staff scheduling research is to cooperate both with a problem owner and with a third-party vendor. In addition, an academic should consider not to work with user interfaces, financial management links, customer reports, help desks, etc. Instead, one should concentrate on modeling issues and algorithmic power.

It is apparent that a profound understanding of the relevant requests and requirements presented by customers is a prerequisite for implementing and solving real-world staff scheduling problems. The implementation should present a wide variety of real-world constraints and be tractable enough to enable the addition of new constraints. It is important to concentrate on the acceptance and satisfaction of both the staff and the personnel managers. Despite the fact that the algorithm should be as robust as possible, no parameter tuning should be expected from the end-users. On the other hand, it should be possible for the end-users to influence different aspects of the algorithm, like weighting between constraints or limit running times, if he/she wishes to.

It should be noted that it is difficult to incorporate the experience and expertise of the personnel managers into a staff scheduling system. Personnel managers often have extremely valuable knowledge, experience and detailed understanding of their specific staffing problem, which will vary from company to company. To formalize this knowledge into constraints is not an easy task. Still, we believe that the model given in this section builds up a solid foundation for staff scheduling scenarios.

The schedules in the model are non-cyclic and can be either continuous or discontinuous. The employees' total working time may vary depending on the employee. Each shift has a fixed start and end time (for example 08:00 – 16:00). Shifts may vary in length and can overlap. Shifts can be classified by shift types which can be used to balance the working times at different times of the day between the employees. A shift is composed of a number of tasks. Each employee has competences that enable her/him to carry out certain tasks. Furthermore,

each employee has preferences over certain tasks, certain shifts and shift types, and certain days-off.

In some cases the most important goal is to minimize understaffing and overstaffing. Low-quality rosters can lead either to an undersupply of employees with a need to hire part-time employees or an oversupply of employees with too much idle time ~~both implicating a loss of business~~. Furthermore, it is very important to pay attention to employee requests. Kellog and Walczak [38] report that any academic model that does not include some opportunity for preference or self-scheduling will probably not be implemented. For example, nurses use complex decision-making skills when selecting their personal schedules. In addition, our experiences have shown that personal managers can generate remarkably good schedules without optimization tools. Therefore, academic research should explore versatile ways to support preference and self-scheduling.

The overall objective is to meet daily staffing requirements and personal preferences at minimum penalty without violating work contracts and government regulations. The presented framework makes no strict distinction between hard and soft constraints; that will be given by the instances themselves. The goal for an instance is to find a feasible solution that is most acceptable for the problem owner. That is, a solution that has no hard constraint violations and that minimizes the weighted sum of the soft constraint violations. The weights will also be given by the instances themselves. Still, one should bear in mind that an instance is usually just an approximation of practice. In reality, hard constraints can turn out to be soft, if necessary, while giving weights to the soft constraints can be difficult.

We classify the constraints into coverage, regulatory and operational requirements, and operational and personal preferences as given in [39]. The coverage requirement ensures that there are a sufficient number of employees on duty at all times. The regulatory requirements ensure that the employee's work contract and government regulations are respected. The personnel's requests are very important and should be met as far as possible; this leads to greater staff satisfaction and commitment, and reduces staff turnover.

The following constraints can be used as a framework for modeling. An academic researcher can examine these issues together with an industry software vendor:

Coverage requirement

- (C1) An employee cannot be assigned to overlapping shifts.
- (C2) A minimum number of employees of particular competences must be guaranteed for each shift or each timeslot
- (C3) A maximum number of employees of particular competences cannot be exceeded for each shift or each timeslot
- (C4) A balanced number of surplus employees must be guaranteed in each working day

Regulatory requirements

- (R1) The required number of working days, working hours and days-off within a timeframe must be respected
- (R2) The required number of holidays within a timeframe must be respected
- (R3) The required number of free weekends (both Saturday and Sunday free) within a timeframe must be respected
- (R4) The minimum rest time within a timeframe must be respected
- (R5) The minimum time gap of rest time between two shifts must be respected
- (R6) The number of special shifts (such as union steward duties and training sessions) for particular employees within a timeframe must be respected
- (R7) Employees cannot work consecutively for more than k days (the maximum length of a work stretch)
- (R8) Some employees cannot work on weekends or during specific hours of the day
- (R9) The maximum number of shifts in a single day must be respected

Operational requirements

- (O1) An employee can only be assigned to a shift he/she has competence for
- (O2) At least k working days must be assigned between two separate days-off

- (O3) An employee cannot be assigned to more than k weekend days within a timeframe
- (O4) An employee cannot be assigned to more/less than k shifts of a given type within a timeframe
- (O5) An employee assigned to a shift type t_1 must not be assigned to a shift type t_2 on the following day (certain stints are not allowed)
- (O6) An employee must be assigned to a particular shift or on-duty or off-duty on a particular day or during a particular timeslot

Operational preferences

- (E1) Single days-off should be avoided
- (E2) Single working days should be avoided
- (E3) The maximum length of consecutive days-off is k
- (E4) A balanced assignment of single days-off and single working days must be guaranteed between the employees
- (E5) A balanced assignment of different shift types must be guaranteed between the employees
- (E6) A balanced assignment of different tasks must be guaranteed between the employees
- (E7) A balanced assignment of weekdays must be guaranteed between employees
- (E8) Assign or avoid a given shift type before or after a free period (days-off, vacation)
- (E9) Assign as many wanted, and avoid as many unwanted, stints as possible

Personal preferences

- (P1) Assign or avoid assigning given employees to the same shifts
- (P2) Assign a requested day-on or avoid a requested day-off
- (P3) Assign a requested shift or avoid an unwanted shift
- (P4) Assign a shift (work) in a requested timeslot or assign no shift (free) to a requested timeslot.

Often, an employee cannot be assigned to more than one shift per day, although we have seen applications where more than one shift per day is allowed. The definition of constraint C1 allows two or more shifts to be assigned provided they do not overlap. Instead of using shifts in C2 and C3, the minimum and maximum number of on-duty staff can be assigned to timeslots. Quite often, a company has more employees working than are needed to cover the minimum number of employees each working day. The surplus employees are used to cover the expected sick leaves and other no-shows (see constraint C4).

Constraints R1-R9 can be different from one employee to another, based on the employees' contracts. The number of working days, working hours and days-off in constraint R1 need not be exact. We can allow minor deviations to working hours over the planning horizon because it might be impossible to meet the number of hours specified in the contract during the planning horizon using the predefined shifts. For example, the number of working hours could be given as a min-max or a $\pm p$ % range to fit this constraint. Shortage or excess is typically handled by the user in the next planning period.

As an example of a work contract and operational requirements and preferences, consider the following. An employee should have exactly nine days-off in every four-week timeframe starting from the beginning of the planning horizon (R1). He/she should have at least one weekend off (R3). The number of working days on weekends cannot exceed four (O3). An employee cannot work consecutively for more than six days (R7). At least two working days must be assigned between two separate days-off (O2). Single days-off and single working days should be avoided (E1 and E2). The maximum length of consecutive days-off is three (E3). Some employees have a work contract that does not involve weekend work (R8).

The working days and shifts in the sample case could be built up using the following rules. The number of working hours per week is 36 hours (R1). At least eleven hours of rest are required during any period of twenty-four consecutive hours (R4). The maximum number of night shifts is three in every four-week timeframe (O4). A night shift has to be followed by at least fourteen hours of rest (R5). An employee assigned to a late or night shift must not be assigned to an early shift the following day (O5). Furthermore, a night/early shift should be

avoided before/after a free period (E8). Note, that the special shifts (R6) can be treated as regular shifts.

Each employee has competences (qualifications and skills) that enable him/her to carry out certain tasks. A shift may require the employee assigned to it to possess one or more competences (O1).

Constraint E9 can be used to model wanted and unwanted shift patterns. The user can input a number of wanted and unwanted stints (patterns). Examples of unwanted stints are D-E-D (Day-Evening-Day) or E-D-E; frequently changing the time of day is stressful. The balanced assignments in constraints E4-E7 can be guaranteed either by using a given min-max range or the range can be expressed as the maximum percentage deviation from the mean.

Constraint P1 can be used in cases when certain employee groups need to work together on the same shifts. Another case could be when a married couple should have the same working hours.

Constraint O6 together with constraints P2, P3 and P4 build up the basis for preference scheduling. The O6 constraints cover any preferences that must be satisfied while the P2, P3 and P4 constraints represent the requests from or wishes of the employees. An efficient and effective preference scheduling relies on having a system whereby the employees can see the requirements, available shifts, an overview of the currently scheduled hours and an easy method of checking the regulations or possible rule violations caused by proposed changes to the schedule. If the schedule created by the employee preferences is close to satisfying the constraints and the coverage demand, it becomes easier to create a schedule that is both feasible with regard to the scheduling constraints and includes most of the requests and preferences. Therefore, it is beneficial for both the employer and the employee to have a good system for preference scheduling. A preference scheduling system that focuses on employee satisfaction should allow each employee a limited number of strict requests of type O6 to schedule days off for things such as doctor's appointments, birthdays and so on, while the number of requests of type P2, P3 and P4 should be enough for the employees to be able to completely schedule all their working hours.

In the case of staff scheduling over several entities (e.g. two departments in a hospital), constraints R1 and O6 may be used to model the days and working time that are "locked" by the other entity (department). For example, Annie is working 70 % for her home department and 30 % for another department. In her home department her 70 % position is modeled by R1 (reduced numbers compared to a 100 % position) and the shifts assigned by the other department is modeled by O6 as pre-assigned days off. It might be necessary to also use some additional constraints: assume for instance that Annie is working a night shift in the other department on Tuesday (starting Monday 23:00 and ending Tuesday at 7:00). Tuesday is then a day off at her home department (O6). She most probably cannot work an evening shift in her home department on Monday. This can be modeled using P4 or O6 by assigning a free period in her home department on Monday evening.

Finally, in cases when days-off are scheduled separately prior to staff rostering, the fixed days-off can be preassigned for the staff rostering phase using constraint O6. The number of holidays within a timeframe can also be preassigned (R2) or the holidays can be treated as days-off.

At the beginning of this section we stated that academics should consider allying with industry software vendors. The other way around also holds. However, Kellog and Walczak [38] give several reasons why developers of commercial products rarely consult academics. For academics, it is usually more important to make publications than to make business. This has a consequence that the scope of the models academics create seems to be relatively small. Although the staff scheduling models developed by academics solve the problem instance at hand, they may fall short of meeting the complex needs of the customers. Academic solutions are often not only computer and platform dependent, but can also use commercial mathematical programming solvers for linear programming, mixed integer programming and constraint programming. However, we believe that the recent advancements made in the staff scheduling research are rapidly closing the research-application gap.

We end this section by summarizing some other important factors that real-world staff scheduling research should consider. Based on our experience, we believe that in addition to being capable of presenting a wide variety of real-world constraints, a staff scheduling system should

- generate just a few, clearly different solutions to choose from,
- allow users to specify the importance of requests and requirements,
- minimize the scheduling time,
- run on any modern computer with any operating system,
- not use third-party mathematical software packages with expensive licensing policies and be integratable with existing industry software.

4 Artificial Benchmark Instances

Researchers quite often only solve some special artificial cases or one real-world case. The strength of random test instances is the ability to produce many problems with many different properties. Still, they should be sufficiently simple for each researcher to be able to use them in their test environment. The strength of practical cases is self-explanatory. However, an algorithm performing well on one practical problem may not perform well on another practical problem, which is why we present a collection of test instances for both artificial and real-world cases. We start with artificial cases.

ID	n	w	d	h	s_{max}	c_{max}	E1+E2	O5+E8	r	E7	sol
1	50	1	0	56							64
2	100	2	0	112							0
3	56	4	10	144	6	4	x				0
4	56	4	10	144	6	4	x	x			0
5	56	4	10	144	6	4	x	x	9		0
6	15	3	7	96	5	3	x				0
7	112	4	8	160	6	3	x			x	0
8	112	4	8	160	6	3	x	x		x	0
9	112	4	8	160	6	3	x	x	9	x	1

Table 2: Nine staff scheduling test instances (n = the total number of employees, w = the length of a planning horizon in weeks, d = the number of days-off in the planning horizon (R1), h = the number of working hours per employee within the planning horizon (R1), s_{max} = the maximum length of a work stretch (R7), c_{max} = the maximum length of consecutive days-off (E3), E1+E2 = no single days-off and working days, O5+E8 = forbidden stints (N+M, N+D, N+DO), r = minimum rest time between two working days (R5), E7 = the number of working days between employees on any weekday must not differ by more than 3). The list of the shifts for the test instances can be found in (Ásgeirsson et al. 2010). The solutions (sol) have been found using the algorithm described in (Nurmi and Kyngäs, 2011).

Table 2 shows nine staff scheduling test instances. The instances comply with the model presented in Section 3. The number of employees (n) varies between 50 and 112, and the length of the planning horizon (w) between 1 and 4 weeks. In each instance, the sum of the shift lengths is exactly the same as the total working hours of the employees (see constraint R1 in Section 3). The number of working hours per employee within the planning horizon (h) is the same for all employees.

The first two instances do not include days-off. For the other instances, the maximum length of a work stretch (R7) and the maximum length of consecutive days-off (E3) are given. Their minimum lengths are two (E1+E2).

In four instances, an employee assigned to a night shift must not be assigned to a morning or day shift (O5) the following day, or have a day-off (E8) in the following day. In two of these instances, the minimum rest time between two working days is given (R5).

Finally, in the last three instances, the number of working days between employees on any weekday must not differ by more than three (E7). The list of shifts for the test instances can be found in [40]. The challenge is to find a feasible (no hard constraint violations) solution that minimizes the number of soft constraint violations. All but two constraints are hard. The two soft constraints are:

- each employee should have exactly h working hours (two violations for each minute above h and one violation for each minute below h) (R1)
- certain stints are forbidden (one violation for each such case) (O5+E8).

We were able to find the optimum zero solution for seven of the test instances; we do not know the optimum value for the other instances. We hope these test instances will help researchers to test the implementation value of their solution methods.

5 Real-World Benchmark Instances

The three real-world instances introduced in this section are based on cases we have solved. The understanding of the two issues raised in the beginning of Section 3 has significantly contributed to the quality of the solutions. The instances have been simplified and modified in order to make it easier for researchers to use them in their test environment. The instances are derived from various lines of business and industry in Finland, Iceland and Norway. The instances and list of shifts can be found in [40].

A Finnish bus transit company has 77 bus drivers. The length of the planning horizon is two weeks. The days-off for the planning horizon are given - they have been scheduled separately prior to staff rostering. The fixed days-off can be preassigned for the staff rostering using constraint O6. The number of working days varies between 2 and 10. The average working day is 8 hours. The total number of shifts is 329 per week. The shift lengths vary between 4 hours 15 minutes and 11 hours 14 minutes. An average shift length is 8 hours and 1 minute.

The problem is highly compact. The total number of working days in the planning horizon is only two less than the total number of shifts. Only two Sundays of one employee or one Sunday of two employees will be left empty. Furthermore, the total sum of the working minutes in all the shifts is only sixteen less than the total number of working minutes to be scheduled to the employees.

The challenge is to find a feasible solution that minimizes the number of the following soft constraint violations:

- each employee should have exactly $8n$ working hours, where n is the number of working days for the employee (two violations for each minute above $8n$ and one violation for each minute below $8n$) (R1)
- a night shift should not be assigned before a day-off (ten violations for each such case) (E8).

We believe that mathematical programming techniques on their own, are in many cases too rigid to deal with the multiple and often changing, objectives and goals of the real-world staff scheduling. This first problem was solved using the population-based cooperative local search method described in [41]. We were able to find a solution with 20 soft constraint violations. Remember that the total sum of the working minutes in all the shifts is 16 less than the total number of working minutes to be scheduled to the employees, that is, the theoretical optimum value is 16 soft constraint violations.

An Icelandic call center has 92 employees that must be scheduled in 30 minute timeslots over six weeks. Each day has around 210 possible shifts, ranging from 4 hours up to 11 hours, and the minimum and maximum number of on-duty staff is given for every timeslot

in the planning period. A full-time position requires 38 hours per week, but some employees are working half-time, 19 hours per week. The hard constraints of the problem are: one shift per day at most (R9); the employees must get at least one in every four weekends off (R3) and each employee must get at least 11 consecutive hours of rest in any 24 hour period (R4); the maximum number of consecutive working days is 6 (R7) and the maximum number of working hours in each 24-hour period is 9 hours (R1). Any requests for time off or vacations are treated as hard constraints (P2, P4), but each employee has only a limited number of such requests for each planning period. The employees can also have limitations on working hours included in their contracts, such as no night shifts or no work on weekends (R8). Furthermore, some employees have meetings or other duties that cannot be modified (O6) and some employees are included in the instance but are not part of the rostering process - i.e., their schedule must be kept unchanged (O6).

The soft constraints are: the coverage requirements (C2, C3) that are given for every timeslot should be satisfied; the total number of working hours over the planning period for each employee should be within a given range, usually ± 8 hours (R1). The problem input includes requests for shifts or working hours from the employees (P3, P4), such that most of the employees have requested enough hours to fulfill their working hour requirement. The goal is to find a schedule that fulfills all the hard constraints while minimizing a weighted sum of the soft constraint violations.

This case was solved using multiple-stage greedy local search, with different operations and objectives for each stage [17]. The total penalty due to soft constraints violations was 3616.

A Norwegian hospital ward has 20 employees that must be scheduled over four weeks. There is a fixed manpower plan consisting of nine different shift types, divided into three different categories (Day – six shift types; Evening – two shift types; Night – one shift type). In total, 78 shifts must be scheduled every week. The average weekly working time is 28.5 hours, but the contracts vary between 17.75 hrs/week (50 % part-time position) to 35.5 hrs/week (full time position). The hard constraints of the problem are: one and only one shift per day (R9) (shifts are assigned to the day they start); the employees must work one weekend each, and which one is given by the input (O6); the cover specified in the manpower plan must be covered exactly; the working hours over the planning horizon must not deviate by more than ± 2 % from the contracted amount (i.e., 142 hours. for a full-time position) (R1); some shifts cannot be followed by other shifts (O5) because there must be a minimum time between on-duties (R5). Also, a maximum weekly working time (R1) and a weekly continuous free period (R4) are hard constraints. The challenge is to minimize the weighted penalty associated with the soft constraints: a maximum number of consecutive day shifts (3), evening shifts (2) and night shifts (3), and a maximum five consecutive working days (R7). Furthermore, there should be no single day shifts or night shifts (E2), each employee should have a maximum of five evening shifts (O4) and a min/max of 2/6 night shifts (O4) over the planning horizon. Finally, single days-off should be avoided (E1).

This case has been solved using a hybrid approach between constraint programming and variable neighborhood descent in an iterated local search framework [42]. The best solution value found was 4.48.

6 Conclusions and Future Work

We defined an implementation-oriented staff scheduling framework. We believe that a considerable number of real-world staff scheduling scenarios can be modeled using the constraints presented in this paper. In addition, we hope the presented modeling issues assist academics in getting their research results implemented into commercial systems. This research has contributed to better systems for our industry partners. A set of artificial and real-world instances derived from the actual problems solved for various companies were presented. We have published the best solutions we have found. We invite the staff scheduling community to challenge our results. We believe that the instances will help researchers to test

the implementation value of their solution methods. The instances are available online. We are currently modeling the instances presented in this paper using the xml-based modeling format introduced and managed by Tim Curtois [43]. In addition, we will present the other real-world instances we have been recently working with.

References

1. Garey M.R. and Johnson D.S., *Computers and Intractability. A Guide to the Theory of NP Completeness*. Freeman (1979)
2. Bartholdi, J.J., "A Guaranteed-Accuracy Round-off Algorithm for Cyclic Scheduling and Set Covering", *Operations Research* 29, pp 501–510 (1981).
3. Tien J. and Kamiyama A., "On Manpower Scheduling Algorithms", *SIAM Rev.* 24(3), pp. 275–287 (1982).
4. Lau, H. C., "On the Complexity of Manpower Shift Scheduling", *Computers and Operations Research* 23(1), pp. 93-102 (1996).
5. Kragelund L. and Kabel T., *Employee Timetabling. An Empirical Study*. Master's Thesis, Department of Computer Science, University of Aarhus, Denmark (1998).
6. Fukunaga, A., Hamilton, E., Fama, J., Andre, D., Matan, O. and Nourbakhsh, I., "Staff scheduling for inbound call and customer contact centers", *AI Magazine* 23(4), pp 30-40 (2002).
7. Marx, D., "Graph coloring problems and their applications in scheduling", *Periodica Polytechnica Ser. El. Eng.* 48, pp. 5–10 (2004).
8. Dantzig, G.B., "A comment on Edie's traffic delays at toll booths", *Operations Research* 2, pp 339–341 (1954).
9. Alfares, H.K., "Survey, categorization and comparison of recent tour scheduling literature", *Annals of Operations Research* 127, pp. 145-175 (2004).
10. Ernst, A. T., Jiang H., Krishnamoorthy, M., and Sier, D., "Staff scheduling and rostering: A review of applications, methods and models", *European Journal of Operational Research* 153 (1), pp 3-27 (2004).
11. Meisels, A. and Schaerf, A., "Modelling and solving employee timetabling problems", *Annals of Mathematics and Artificial Intelligence* 39, pp. 41-59 (2003).
12. Burke, E.K., De Causmaecker P., Petrovic S. and Vanden Berghe G., "Variable neighborhood search for nurse rostering problems". In Resende and de Sousa, Editors, *Metaheuristics: Computer Decision-Making*, Kluwer, pp. 153–172 (2004).
13. Dowling, D., Krishnamoorthy, M., Mackenzie, H. and Sier, D., "Staff rostering at a large international airport", *Annals of Operations Research* 72, pp 125-147 (1997).
14. Beer, A., Gaertner, J., Musliu, N., Schafhauser, W. and Slany, W., "Scheduling breaks in shift plans for call centers". In *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling*, Montréal, Canada (2008).
15. Stolletz, R., "Operational workforce planning for check-in counters at airports". *Transportation Research Part E* 46, pp. 414-425 (2010).
16. Lusby T., Dohn A., Range T. and Larsen J., "Ground Crew Rostering with Work Patterns at a Major European Airlines". In *Proc of the 8th Conference on the Practice and Theory of Automated Timetabling (PATAT)*, Belfast, Ireland (2010).
17. Ásgeirsson, E. I., "Bridging the gap between self schedules and feasible schedules in staff scheduling". In *Proc of the 8th Conference on the Practice and Theory of Automated Timetabling (PATAT)*, Belfast, Ireland (2010).
18. Bard, J. F., Binici, C. and Desilva, A. H., "Staff Scheduling at the United States Postal Service", *Computers & Operations Research* 30, pp 745-771 (2003).
19. Nurmi K., Kyngäs J. and Post G. "Staff Scheduling for Bus Transit Companies", In *Proc of the International MultiConference of Engineers and Computer Scientists (ICOR'11)*, Hong Kong (2011).
20. Seçkiner, S.U. and Kurt, M., "Ant colony optimization for the job rotation scheduling problem", *Applied Mathematics and Computation* 201(1-2), pp. 149-160 (2008).

21. Elshafei M. and Alfares H., "A dynamic programming algorithm for days-off scheduling with sequence dependent labor costs", *Journal of Scheduling* 11(2), pp 85-93 (2008).
22. Qu R. and He F., "A hybrid constraint programming approach for nurse rostering problems". In Allen, Ellis, and Petridis, Editors, *Applications and Innovations in Intelligent Systems XVI*, Cambridge, UK, pp. 211-224 (2008).
23. Dean J., "Staff Scheduling by a Genetic Algorithm with a Two-Dimensional Chromosome Structure". In *Proc of the 7th Conference on the Practice and Theory of Automated Timetabling*, Montreal, Canada (2008).
24. Burke, E.K., Curtois, T., Qu, R. and Vanden Berghe, G., "A scatter search methodology for the nurse rostering problem", *Journal of the Operational Research Society* (to be published 2010).
25. Remde, S., Cowling, P. I., Dahal, K. P. and Colledge, N., "Exact/Heuristic Hybrids Using rVNS and Hyperheuristics for Workforce Scheduling", In *Proc. of the 7th Evolutionary Computation in Combinatorial Optimization*, Lecture Notes in Computer Science 4446, Springer, pp. 188-197 (2007).
26. Brunner, J.O., Bard, J.F., and Kolisch, R., "Flexible shift scheduling of physicians", *Health Care Management Science*. 12(3), pp 285-305 (2009).
27. Burke, E., De Causmaecker P., Petrovic S., and Vanden Berghe G., "Metaheuristics for Handling Time Interval Coverage Constraints in Nurse Scheduling", *Applied Artificial Intelligence*, pp 743-766 (2006).
28. Goodale, J. and Thompson, G., "A Comparison of Heuristics for Assigning Individual Employees to Labor Tour Schedules", *Annals of Operations Research* 128(1), pp 47-63 (2004).
29. Musliu, N., "Heuristic Methods for Automatic Rotating Workforce Scheduling", *International Journal of Computational Intelligence Research* 2(4), pp. 309-326 (2006).
30. Burke, E.K., Curtois, T., Post, G.F., Qu, R. and Veltman, B., "A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem", *European Journal of Operational Research* 188(2), pp 330-341 (2008).
31. Bard, J. and Purnomo H., "Hospital-wide reactive scheduling of nurses with preference considerations", *IIE Trans.* 37(7) 589–608 (2005).
32. Beddoe, G.R., Petrovic, S. and Li, J., "A Hybrid Metaheuristic Case-based Reasoning System for Nurse Rostering", *Journal of Scheduling* 12, pp 99–119 (2009).
33. Bilgin, B., De Causmaecker, P., Rossie, B. and Vanden Berghe G., "Local Search Neighbourhoods to Deal with a Novel Nurse Rostering Model". In *Proc. of the 7th Int. Conf. on the Practice and Theory of Automated Timetabling*, Montréal, Canada (2008).
34. Diaz, T., Ferber, D., deSouza C. and Moura A., "Constructing nurse schedules at large hospitals", *Internat. Trans. Oper. Res.* 10(3), pp. 245–265, (2003).
35. Kawanaka, H., Yoshikawa T., Shinogi T. and Tsuruoka S., "Constraints and search efficiency in nurse scheduling problem". In *Proc. Internat. Sympos. Comput. Intelligence Robotics Automation* 1, pp. 312–317 (2003).
36. Meyer auf'm Hofe, H., "Solving rostering tasks by generic methods for constraint optimization". *Internat. J. Foundations Comput. Sci.* 12(5), pp. 671–693 (2001).
37. Van Wezel, W. and Jorna R., "Scheduling in a generic perspective: Knowledge-based decision support by domain analysis and cognitive task analysis", *Internat. J. Expert Systems* 9(3), pp. 357–381 (1996).
38. Kellogg D.L. and Walczak S., "Nurse Scheduling: From Academia to Implementation or Not", *Interfaces* 37(4), pp. 355-369 (2007).
39. Bellanti, F., Carello, G., Della Croce F., and Tadei R., "A greedy-based neighborhood search approach to a nurse rostering problem", *European Journal of Operational Research*, 153(1), pp. 28–40 (2004).
40. Ásgeirsson, E.I., Kyngäs J., Nurmi, K. and Stølevik, M. (Last update 30.5.2011). "Implementation-Oriented Staff Scheduling Problem" [Online], Available: <http://www.samk.fi/iiossp>.

41. Nurmi K. and Kyngäs J., “Improving the Schedule of the Finnish Major Ice Hockey League”. In *Proc. of the 2nd International Conference on the Mathematics in Sport*, Groningen, Netherlands (2009).
42. Stølevik, M, Nordlander T. E., Riise, A. and Frøyseth, H., “An efficient hybrid approach that solves real world nurse rostering problems, using constraint programming and variable neighborhood descent” (preprint submitted to Engineering Applications of Artificial Intelligence 2010).
43. Curtois, T. (Last update August 2010). “Staff Rostering Benchmark Data Sets” [Online]. Available: <http://www.cs.nott.ac.uk/~tec/NRP/>

Dynamic Data Structures for Taskgraph Scheduling Policies with Applications in OpenCL Accelerators

Jakub Mareček · Andrew J. Parkes ·
Edmund K. Burke · Robert Elliot ·
Hedley Francis · Anton Lokhmotov

Abstract OpenCL is an emerging open framework for parallel programming in heterogeneous systems. OpenCL accelerators need to schedule the execution of submitted jobs with no (or only very imprecise) estimates of execution times, but respecting dependencies among them, which are given in the form of directed acyclic graph. This problem is known as stochastic taskgraph scheduling, stochastic scheduling with precedencies, or stochastic scheduling with data dependencies.

We study the complexity of implementing out-of-order static scheduling policies, which approach optimality in the long run, under certain simplifying assumptions. We present a simple data structure allowing for the “what next” query of a number of out-of-order taskgraph scheduling policies to be answered in time $O(1)$, while vertices can be added in time $O(1)$.

1 Introduction

In stochastic taskgraph scheduling, policies which run jobs with the largest sum of expected processing times along an out-going path in the dependency graph first perform particularly well. Papadimitriou and Tsitsiklis [20] have shown that, under certain restrictions, they are asymptotically optimal. This is not particularly surprising, as they correspond to the well-known critical path heuristics [12] in the off-line case. An important question remains, though: how efficiently can one implement such policies?

The implementation of “longest-path” policies requires a dynamic data structure for maintaining a criterion related to out-going paths in a vertex-weighted directed acyclic graph (DAG), subject to the insertion of a sink with adjacent edges and deletion of a source that maximises a certain criterion, together with adjacent edges. The data structure is parametrised by the criterion, which is one of the following: the number of

Jakub Mareček is the contact author. E-mail: jakub@marecek.cz.

Jakub Mareček · Andrew J. Parkes · Edmund K. Burke
School of Computer Science, The University of Nottingham, Nottingham NG8 1BB, UK
E-mail: { jxm, ajp, ekb }@cs.nott.ac.uk

Robert Elliot · Hedley Francis · Anton Lokhmotov
ARM Ltd., 110 Fulbourn Road, Cambridge CB1 9NJ, UK
E-mail: { Robert.Elliot, Hedley.Francis, Anton.Lokhmotov } @arm.com

Table 1 Data structures for out-of-order static scheduling policies and the corresponding upper bounds on the complexity of key operations: worst case analysis of a trivial use of linked lists compared to the input and output model analysis [24] of the proposed data structure. Note n is the number of vertices, $|\delta|$ is the number of vertices on the affected longest paths, and $||\delta||$ is the number of vertices in the neighbourhoods of the vertices along the affected longest paths, in all three cases in the resulting graph, and $Q(n)$ is the complexity of insertion and extraction of an element from a priority queue with n elements, which is $\sqrt{\log \log n}$ (amortised) for melding priority queues [17].

Operation	Linked lists	This paper
merge (H, C)	$O(C n)$	$O(C \log C + \delta Q(\delta) + \delta)$
popVertex	$O(n^2)$	$O(\delta Q(\delta) + \delta)$
“What next” query	$O(n^2)$	$O(1)$
Sink insertion	$O(1)$	$O(1)$
Edge-to-sink insertion	$O(n)$	$O(\delta Q(\delta) + \delta)$
Source deletion	$O(n)$	$O(\delta Q(\delta) + \delta)$

jobs along the longest path in the subgraph reachable from the source or the number of successors in the reachable subgraph, possibly weighed by run-time estimates in either case. The required operations are:

- **popVertex**: delete the vertex with no in-going edges that maximises the criterion.
- **merge**(H, C): add (possibly smaller) DAG H to the current (possibly larger) DAG G . There will be no edges from H to G , but there are $|C|$ “cross” edges from G to H passed separately as C .

We also study the following special cases of the two operations:

- “What next” query, picking the (source) vertex with no in-going edges, which maximises one of the criteria mentioned above.
- Deletion of a (source) vertex with no in-going edges, together with all adjacent out-going edges.
- Insertion of a (sink) vertex and a number of in-going edges.

As far as we know, no dynamic data structure supporting even the special cases of the two operations efficiently has been studied previously. There are, however, dynamic data structures for maintaining topological order in DAGs [10] subject to edge insertion and deletion, dynamic data structures for maintaining the length of the shortest or longest in-going path in a DAG [11] subject to edge insertion and deletion, and numerous data structures for maintaining related information in undirected trees subject to a wide range of operations [2, 8, 26]. In Section 3, we present the data structure, together with an analysis in the input and output model of Ramaling and Reps [24], summarised in Table 1. First, however, we introduce the motivating problem in the Section 2, with the related work on taskgraph scheduling policies summarised in Subsection 2.4.

2 The Motivating Problem

OpenCL is being developed by AMD, Apple, ARM, Intel, Motorola, Nokia, NVidia, Qualcomm, Samsung, Sony, Sun, Texas Instruments, and a number of others, as an open, royalty-free standard for parallel programming in heterogenous systems. The OpenCL specification envisions the bulk of demanding computations being handled by an OpenCL accelerator, rather than the central processing unit (CPU). The OpenCL

accelerator maintains a “system queue”, which describes what jobs are to be executed and what are the acyclic dependencies among them. This work load comes from a number application, but only indirectly. Each application can have multiple pieces of OpenCL-accelerated code, or “kernels”, and a number of “local queues” each. Each “kernel” can be executed multiple times and we denote each execution as a “job”. (This is important, as one may expect multiple execution of the same kernel to have similar properties, including run-time.) Jobs are first submitted to the “local queue”, where the application can specify further acyclic dependencies among the jobs in the queue, as well as dependencies on the jobs in the system queue. At the point when a “local queue” is “flushed”, the scheduler takes over the control. We assume that at most one “kernel” can be run on the OpenCL accelerator at any point in time, which is realistic in embedded applications. If the scheduling was sequential and in-order, jobs in a local queue could be sorted topologically after its flushing, and dependencies between jobs could be disregarded in the “system queue”. When the scheduler requires parallel or out-of-order execution, however, acyclic dependencies between jobs need to be stored and checked, before a job is run, or maintained otherwise. Considering that the accelerators are massively parallel and out-of-order execution may be required to accommodate latency constraints, the latter is the case. Notice that, unlike in games consoles where the application and hardware specification is known [3], one cannot easily pre-compute a schedule of an unknown workload. Note that thousands or millions of jobs may well need to be processed per second, and the overhead due to the scheduler needs to be kept to a minimum, especially in battery-powered applications. One, hence, needs a fast implementation of a policy for stochastic task-graph scheduling.

2.1 The Problem

In stochastic task-graph scheduling, we assume that there exists an unknown vertex-weighted directed acyclic graph, “taskgraph”, with countably many nodes. Vertices of the taskgraph correspond to jobs and there is an edge between vertices u, v , if and only if job v must be run only after completion of job u . The vertex weights in the taskgraph correspond to run-times of the jobs. Jobs (vertices) are revealed in batches, together with some estimates of their run-times. Jobs, which do not have any dependencies, are called “available”. There are also m identical machines, each of which can process any of the available jobs.

A policy is a rule for deciding, which of one of the available jobs is to be run on a machine. In particular, we are interested in the families of:

- non-delay policies, where a job is executed, whenever there are jobs available and there is capacity to process at least one of them
- non-anticipative policies, where no assumptions are made about jobs arriving in the future
- non-preemptive policies, where a job is run until completion, whenever it is run.
- preemptive policies with preemption cost c_p , where a job can be stopped at any point and resumed after a preemption routine, whose runtime is c_p .

In practice, there are considerable preemption costs in preemptive policies. This preemption routine needs to save a great number of registers and may increase the number of cache misses, which may be comparable to running the average job, and hence make

preemptive policies with realistic preemption costs very close to non-preemptive policies.

In the long-run horizon, we study the throughput and (discounted) weighted makespan of policies. More formally, we focus on:

- the weighted throughput of the system, given by policy π :

$$J(\pi) = \limsup_{t \rightarrow \infty} \frac{1}{t} \sum_{q \in Q} w(q) \mathbb{E}[a_q^\pi(t)]$$

- the α -discounted weighted makespan of the system, given by policy π :

$$K(\pi) = \sum_{q \in Q} \sum_{i=1}^{\infty} w(q) \mathbb{E}[e^{-\alpha C_i(q)}]$$

where a_q is the number of jobs from queue q completed by time t , and $C_i(q)$ is the completion time of i th job in queue q , both of which are well-defined random variables, and $0 < \alpha \leq 1$ is the discount rate.

Given an objective function f , input σ , and the optimum of f on σ , $OPT_f(\sigma)$, the asymptotic approximation (performance, competitive) ratio of policy π of is:

$$R_f^\infty(\pi) = \limsup_{n \rightarrow \infty} \left\{ \frac{\pi_f(\sigma)}{OPT_f(\sigma)} \mid OPT_f(\sigma) = n \right\}.$$

Within a family of policies P , policies with approximation ratio:

$$R_f^\infty = \inf_{\pi \in P} R_f^\infty(\pi).$$

are asymptotically optimal.

2.2 The Complexity

The problem of scheduling the stochastic OpenCL Task System and its precisely revealed deterministic snapshot, called the Precisely Revealed OpenCL Task System, has been formalised and studied in another paper by the authors [16]. When one formalises the problems, it is easy to see that the problem of deciding the non-preemptive schedulability of a Precisely Revealed OpenCL Task System within a finite time horizon T is $\mathcal{NP}$ -Hard. Indeed, non-preemptive schedulability of jobs with precedencies on two or more machines within a finite time horizon has been on Karp's original list of $\mathcal{NP}$ -Complete problems [9]. Perhaps more interestingly, deciding if there is a non-preemptive policy resulting in priority-weighted throughput larger than k for a (Forgetful) OpenCL Task System in an OpenCL Accelerator is $\mathcal{EXP}^{\mathcal{PTIME}}$ -Hard, if there is a fixed assignment of queues to cores. For closed queuing systems (Forgetful OpenCL Task System), one can use the reduction to $\mathcal{NETWORKOFQUEUES}$ of Papadimitriou and Tsitsiklis [19,21]. For open queuing systems (OpenCL Task System), one needs to prove a similar result for a variant of $\mathcal{NETWORKOFQUEUES}$. The problem is hence hard, independently of any unproven conjectures, such as $P \neq NP$. We can show, however, that there are asymptotically optimal policies, under certain assumptions.

2.3 The Stability

Let us consider the limits of stability of a closely related system. We define the system to be in a “transient state” (“choked”), if the number of jobs waiting in any queue goes to the large limit in the long run. We define the system to be “stable” if for all queues, there is a finite bound on the expected interval between two times when the number of work-groups waiting to be processed is zero. It is clear that the stability depends on the arrival rates of jobs, their processing times, and dependencies between work-groups.

In particular, we are interested in the effects of dependencies between jobs. We assume that job j depends on any enqueued jobs with probability p , independently of any other dependencies. λ^* is the best possible arrival rate, that is the interval between the arrival of two jobs. Then:

Theorem 1 (Hajek [27]) $\lim_{p \rightarrow 0} \lambda^* p = e^{-1}$

Let us now study the $n - m$ work-groups waiting to be processed, out of n work-groups seen so far. $G[m, n]$ is the subgraph of the taskgraph induced by vertices $\{m, m+1, \dots, n\}$. We denote d_{mn} the length of longest path in $G[m, n]$. Further, $\beta_n = \mathbb{E}[d_{1n}]/n$. Then:

Theorem 2 (Tsitsiklis et al. [27]) *The limit $\lim_{n \rightarrow \infty} (d_{mn}/n)$ exists almost surely. Let us use β^* to denote the limit for any m where it exists. If $\lambda < 1/\beta^*$, the system is stable.*

The results below are not conditional on the system being in a stable state, but draw a distinct inspiration from the dependency of the performance on the length of the longest path in the data dependency graph.

2.4 A Policy and Conditions of its Asymptotic Optimality

Let us now consider the non-delay non-preemptive policy scheduling the available job that corresponds to the root in the taskgraph maximising the numbers of jobs along the longest path in the subgraphs reachable from the root, or “level”, whenever there are available jobs and a core becomes available. Let us assume that:

- there are jobs with independent identically distributed processing times, drawn from a common binomial or Poisson distribution
- data dependency graph G is a forest of in-trees, and the non-directed counterpart of the taskgraph is hence acyclic.

When we denote these assumptions by an asteriks (*), it has been shown:

Theorem 3 (Papadimitriou and Tsitsiklis [20]) *Processing the job with the highest level (“largest sum of expected processing times along a path in the dependency graph”) first, as soon as any machine is available, is asymptotically optimal with respect to makespan, under certain conditions (*), among non-anticipative non-delay non-preemptive policies and non-anticipative non-delay preemptive policies with zero cost of preemption.*

This means that the makespan achieved by the “largest sum of expected processing times along a path in the dependency graph”-first policy is no larger than the optimal makespan by a factor that goes to one in the large limit of the number of jobs. The proof is based on an exchange argument, due initially to Papadimitriou and Tsitsiklis [20] and extended by Liu and Sanlaville [14,15,15]. Similar techniques had appeared in the proofs of Pinedo and Weiss [22] and Chandy and Reynolds and Bruno [4], which were all restricted to two identical processors. Notice that if there were multiple processors running different jobs and arbitrary restrictions on jobs running on certain servers, the policy would no longer be even asymptotically optimal [25].

A number of important questions remain, however: What is the run-time of the corresponding query and is it offset by the improvements over a trivial (first-come first-served) scheduler? The problem of finding the longest path in either general graphs or digraphs is clearly $\mathcal{NP}$ -Hard, as the HPP [9] is a special case. The corresponding decision problem in directed acyclic graphs is, however, equivalent to finding the shortest path in undirected graph, and hence in $\mathcal{NL} \subseteq \mathcal{P}$. In order to make the answer more precise, we need to present the corresponding data structures.

3 The Fully-Dynamic Data Structure

In Algorithms 1–6, we present dynamic data structures for maintaining both the length of the longest in-coming path (depth) and the length of the longest out-going path (level) in each node of a directed acyclic graph. The approach for maintaining the data is based on the “contraption under gravity” view developed by Dijkstra in the 1960s [7,5,11]: First, we traverse the out-going subgraph (“down”) from the affected vertex along a topological order of vertices, to update the lengths of the longest in-going paths in the sub-graph (depth). Next, we traverse the in-coming subgraph in the reverse direction (“up”) along a topological order of vertices, updating the level as long as it is necessary. This can be visualised as a “contraption” of strings and knots held at the knot corresponding to the affected vertex, updating the depth, holding the bottom most knot, and updating the level. The non-trivial part is the maintenance of the topological order of vertices [10], so as to guarantee no edge needs to be visited twice.

Katriel et al. [11] have analysed the use of a priority queue with the priority being the depth of the vertex in order to obtain the topological order, and applications in maintaining the longest in-coming paths (depth) in directed acyclic graphs subject to edge insertion and deletion. The analysis has been subsequently improved [13,1]. In Algorithms 1–6, we adapt the approach of Katriel et al. [11] to the maintenance of both the length of the longest in-coming path (depth) and the length of the longest out-going path (level) in each node of a vertex-weighted directed acyclic graphs under the operations required in the out-of-order static scheduling policies. Alternatively, one could maintain only the lengths of the longest out-going path (level). This would allow for faster run-times of arbitrary edge insertion, which we do not require, but which would slow down the run-time of **merge** by the all-pairs shortest paths computation.

Let us now analyse the algorithms in the input and output model of Ramaling and Reps [24], using melding priority queues [17]. $|\delta|$ is the number of vertices whose “longest path” to root has changed and $||\delta||$ is the cardinality of the union of neighbourhoods of vertices whose “longest path” to root has changed.

Proposition 1 `insertEdgeUpdateDownstream` runs in time $O(|\delta| + |\delta|Q(|\delta|))$, `insertEdgeUpdateUpstream` runs in time $O(|\delta| + |\delta|Q(|\delta|))$, and `insertEdge` runs in time $O(|\delta| + |\delta|Q(|\delta|))$, where $Q(n)$ is the complexity of insertion and extraction of an element in a priority queue with n elements.

Proof sketch: The run-time complexity of `insertEdge` is dominated by the complexity of `insertEdgeUpdateDownstream`. There, each vertex is inserted and extracted from the queue at most once, and all $|\delta|$ neighbouring vertices need to be checked.

Proposition 2 `merge(A,B)` runs in time $O(c \log c + |\delta|Q(|\delta|) + |\delta|)$, where c is the number of “cross” edges from B to A , δ corresponds to changes in both A and B , and $Q(n)$ is the complexity of insertion and extraction of an element in a priority queue with n elements.

Proof sketch: The complexity of ordering c edges by the sum of the longest paths to the edge in both A and B is $c \log c$. This gives us the order of updates, where there is no vertex updated twice.

Proposition 3 `topVertex` runs in time $O(1)$. `popVertex` runs in time $O(|\delta|Q(|\delta|) + |\delta|)$. `popVertexUpdateDownstream` runs in time $O(|\delta|Q(|\delta|) + |\delta|)$.

Proof sketch: The complexity of `topVertex` is given by the “top” operation of a priority queue [17]. The run-time complexity of `popVertex` is dominated by the complexity of `popVertexUpdateDownstream`, which is $O(|\delta|Q(|\delta|) + |\delta|)$. No vertex is updated twice, but there are $|\delta|$ vertices to be updated and $|\delta|$ neighbouring vertices to be checked.

Notice that for melding priority queues [17], the amortised complexity of insertion and extraction is $Q(n) := \sqrt{\log \log n}$. Both $|\delta|$ and $|\delta|$ can be large in the worst-case, notably linear in the number of vertices and edges in `merge`. This is, however, perhaps inevitable. In the closely related problem of prioritising vertices of a DAG, where each vertex is assigned a priority such that, for each oriented edge (v, w) , $\text{priority}(v) < \text{priority}(w)$, Ramalingam and Reps [23] have shown a lower bound of $\Omega(n \log n)$ operations on the insertion of m edges in a graph on n vertices. There are good reasons [1] to believe that the performance is considerably better in expectation. Also, notice that the `topVertex` runs in constant time, so that the execution of the scheduler need not hinder the execution of the jobs on the accelerator.

4 Conclusions and Open Problems

As far as we are aware, we have presented the first fully-dynamic data structures for implementing certain out-of-order taskgraph scheduling policies, with an application in the design of drivers for OpenCL accelerators. Unlike the present best data structures for similar operations on (undirected) trees, such as top trees [2], we do not perform updates lazily, which may well leave space for improvement:

- Can top trees be extended to directed acyclic graphs? Top trees are, in turn, based on data structures proposed by Tarjan et al. [8, 26] earlier. Could those be extended?
- Are there lower bounds on the complexity of `popVertex` and `merge`?

There are also great many questions related to the scheduling policies left open:

- What is the broadest class of graphs for which the studied static scheduling policies are asymptotically optimal? This seems to be one of the most important open questions in scheduling.
- What are the limits of stability in realistic models of queuing networks with dependencies? Consider jobs partitioned into groups. How does the stability threshold change, when there are no intra-group dependencies and jobs within each share the dependencies? How does the stability threshold change, when the probability of dependence of group a on a group b is inversely proportional to the difference in their release dates $r_a - r_b$?
- How much could one be benefit from pilot runs improving the run-time estimates? Could we decide how many pilot runs to perform, based on some measures of quality of the estimates obtained so far?
- What are the benefits of dynamic scheduling policies, such as the multi-mode multi-armed bandits [28, 18]? Notably, could they be used to integrate power management considerations? Could the data structure be extended to accommodate such “indices”?
- What are the benefits of “taskgraph prediction”? In a rather different setting, Chekuri et al [6] have shown that the longest path scheduling based on the known dependency graph is not optimal, when we expect changes of the dependency tree in the future and can make educated guesses about their nature.

There are also several questions specific to accelerators developed by ARM. Considering the results on stability referenced in Section 2.3, however, whatever improved policies there might be, it seems highly likely that they will be require the maintenance of weighted long paths. The research into the dynamic data structures implementing them will hence remain highly relevant.

Acknowledgments The work has been supported by the Industrial Mathematics Internships Programme, funded by EPSRC and ARM Ltd. OpenCL is a trademark of Apple Inc. used by Khronos by permission.

References

1. Ajwani, D., Friedrich, T.: Average-case analysis of online topological ordering. In: Algorithms and computation, *Lecture Notes in Comput. Sci.*, vol. 4835, pp. 464–475. Springer, Berlin (2007). DOI 10.1007/978-3-540-77120-3_41. URL http://dx.doi.org/10.1007/978-3-540-77120-3_41
2. Alstrup, S., Holm, J., Thorup, M., de Lichtenberg, K.: Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms* **1**(2), 243–264 (2005). DOI 10.1145/1103963.1103966
3. Benini, L., Lombardi, M., Milano, M., Ruggiero, M.: Optimal resource allocation and scheduling for the cell be platform. *Ann. Oper. Res.* **184**, 51–77 (2011). 10.1007/s10479-010-0718-x
4. Bruno, J.: On scheduling tasks with exponential service times and in-tree precedence constraints. *Acta Inform.* **22**(2), 139–148 (1985). DOI 10.1007/BF00264227
5. Bulterman, R.W., van der Sommen, F.W., Zwaan, G., Verhoeff, T., van Gasteren, A.J.M., Feijen, W.H.J.: On computing a longest path in a tree. *Inform. Process. Lett.* **81**(2), 93–96 (2002). DOI 10.1016/S0020-0190(01)00198-3
6. Chekuri, C., Johnson, R., Motwani, R., Natarajan, B., Rau, B.R., Schlansker, M.S.: Profile-driven instruction level parallel scheduling with application to super blocks. In: MICRO, pp. 58–67 (1996)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. Second edn. MIT Press, Cambridge, MA (2001)

8. Goldberg, A.V., Grigoriadis, M.D., Tarjan, R.E.: Use of dynamic trees in a network simplex algorithm for the maximum flow problem. *Math. Programming* **50**(3, (Ser. A)), 277–290 (1991). DOI 10.1007/BF01594940
9. Karp, R.M.: Reducibility among combinatorial problems. In: *Complexity of computer computations* (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972), pp. 85–103. Plenum, New York (1972)
10. Katriel, I., Bodlaender, H.L.: Online topological ordering. *ACM Trans. Algorithms* **2**(3), 364–379 (2006). DOI 10.1145/1159892.1159896
11. Katriel, I., Michel, L., Van Hentenryck, P.: Maintaining longest paths incrementally. *Constraints* **10**(2), 159–183 (2005). DOI 10.1007/s10601-005-0554-9
12. Kelley Jr., J.E.: Critical-path planning and scheduling: mathematical basis. *Operations Res.* **9**, 296–320 (1961)
13. Liu, H.F., Chao, K.M.: A tight analysis of the Katriel-Bodlaender algorithm for online topological ordering. *Theoret. Comput. Sci.* **389**(1-2), 182–189 (2007). DOI 10.1016/j.tcs.2007.08.009
14. Liu, Z., Sanlaville, E.: Preemptive scheduling with variable profile, precedence constraints and due dates. *Discrete Appl. Math.* **58**(3), 253–280 (1995). DOI 10.1016/0166-218X(93)E0151-N
15. Liu, Z., Sanlaville, E.: Stochastic scheduling with variable profile and precedence constraints. *SIAM J. Comput.* **26**(1), 173–187 (1997). DOI 10.1137/S0097539791218949
16. Mareček, J., friends: The complexity of scheduling in OpenCL accelerators. Submitted
17. Mendelson, R., Tarjan, R.E., Thorup, M., Zwick, U.: Melding priority queues. *ACM Trans. Algorithms* **2**(4), 535–556 (2006). DOI 10.1145/1198513.1198517
18. Niño-Mora, J.: An index policy for multiarmed multimode restless bandits. In: *ValueTools '08: Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools*, pp. 1–6. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium (2008). DOI <http://dx.doi.org/10.4108/ICST.VALUETOOLS2008.4410>
19. Papadimitriou, C.H., Tsitsiklis, J.: Intractable problems in control theory. *SIAM J. Control Optim.* **24**(4), 639–654 (1986). DOI 10.1137/0324038
20. Papadimitriou, C.H., Tsitsiklis, J.N.: On stochastic scheduling with in-tree precedence constraints. *SIAM J. Comput.* **16**(1), 1–6 (1987). DOI 10.1137/0216001
21. Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of optimal queuing network control. *Math. Oper. Res.* **24**(2), 293–305 (1999). DOI 10.1287/moor.24.2.293
22. Pinedo, M., Weiss, G.: Scheduling jobs with exponentially distributed processing times andintree precedence constraints on two parallel machines. *Oper. Res.* **33**(6), 1381–1388 (1985). DOI 10.1287/opre.33.6.1381
23. Ramalingam, G., Reps, T.: On competitive on-line algorithms for the dynamic priority-ordering problem. *Inform. Process. Lett.* **51**(3), 155–161 (1994). DOI 10.1016/0020-0190(94)00080-8
24. Ramalingam, G., Reps, T.: On the computational complexity of dynamic graph problems. *Theoret. Comput. Sci.* **158**(1-2), 233–277 (1996). DOI 10.1016/0304-3975(95)00079-8
25. Stolyar, A., Yudovina, E.: Systems with large flexible server pools: Instability of “natural” load balancing. *ArXiv e-print* (2010)
26. Tarjan, R.E.: Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. *Math. Programming* **78**(2, Ser. B), 169–177 (1997). DOI 10.1016/S0025-5610(97)00015-4
27. Tsitsiklis, J.N., Papadimitriou, C.H., Humblet, P.: The performance of a precedence-based queueing discipline. *J. Assoc. Comput. Mach.* **33**(3), 593–602 (1986). DOI 10.1145/5925.5936
28. Weber, R.R.: Comments on: Dynamic priority allocation via restless bandit marginal productivity indices. *TOP* **15**(2), 211–216 (2007). DOI 10.1007/s11750-007-0029-9

Algorithm 1 insertEdge($G, G_{UL}, G_{WL}, R, u \rightarrow v$)

```

1: Input: Digraph  $G$ , weighted and unweighted auxiliary structures  $G_{UL}, G_{WL}$ , priority
   queue  $R$ , edge  $u \rightarrow v$  to be added
2: Effect: Updated  $G, G_{UL}, G_{WL}, R$ 
3:  $G = G \cup \{u \rightarrow v\}$ 
4: if has( $R, v$ ) then
5:   remove( $R, v$ )
6: end if
7:  $U = \text{insertEdgeUpdateDownstream}(G_{UL}, u, v)$ 
8:  $U = U \cup \text{insertEdgeUpdateUpstream}(G_{UL}, u, v)$ 
9:  $U = U \cup \text{insertEdgeUpdateDownstream}(G_{WL}, u, v)$ 
10:  $U = U \cup \text{insertEdgeUpdateUpstream}(G_{WL}, u, v)$ 
11: while  $U \neq \emptyset$  do
12:    $u = \text{top}(U)$  //  $u$  for updated
13:   update( $R, u$ )
14: end while

```

Algorithm 2 insertEdgeUpdateUpstream($G_L, u \rightarrow v$)

```

1: Input: Structure  $G_L$ , edge  $u \rightarrow v$  to be added
2: Output: List of updated vertices  $U$ 
3: Effect: Updated structure  $G_L$ 
4:  $U = \emptyset$ 
5:  $Q = \text{queue} \langle \text{WeightedLongestFirst} \rangle(v)$  // forwards
6: while  $Q \neq \emptyset$  do
7:    $v = \text{pop}(Q)$ 
8:   for  $u \in \text{pred}(G_L, v)$  do
9:     if level( $u$ ) < level( $v$ ) + weight( $v$ ) then
10:      level( $u$ ) = level( $v$ ) + weight( $v$ )
11:      insert( $Q, u$ )
12:      insert( $U, u$ )
13:     end if
14:   end for
15: end while
16: return  $U$ 

```

Algorithm 3 insertEdgeUpdateDownstream($G_L, u \rightarrow v$)

```

1: Input: Structure  $G_L$ , edge  $u \rightarrow v$  to be added
2: Output: List of updated vertices  $U$ 
3: Effect: Updated structure  $G_L$ 
4:  $U = \emptyset$ 
5: if  $l(u) + d(u, v) > l(v)$  then
6:    $Q = \text{queue}(\text{WeightedLongestFirst})()$  // forwards
7:    $B = \text{queue}(\text{WeightedLongestFirst})()$  // backwards
8:    $\text{insert}(Q, \langle l(v), v \rangle)$ 
9:   while  $Q \neq \emptyset$  do
10:     $a = \text{extractMin}(Q)$  // a for affected
11:     $l(a) = \max_{x \in \text{pred}(G, a)} l(x) + \text{weight}(a)$ 
12:     $G_L = G_L \setminus \{u \rightarrow a \mid u \rightarrow a \in G_L\}$ 
13:     $G_L = G_L \cup \{u \rightarrow a \mid x \in \text{pred}(G, a) \wedge l(u) + \text{weight}(a) = l(a)\}$ 
14:    if  $|\text{succ}(G, a)| = 0$  then
15:       $\text{level}(a) = 1$ 
16:       $\text{insert}(B, a)$ 
17:       $\text{insert}(U, u)$ 
18:    end if
19:    for  $b \in \text{succ}(G, a)$  do
20:      if  $l(a) + \text{weight}(b) > l(b)$  then
21:         $\text{insert}(Q, \langle l(b), b \rangle)$ 
22:      else
23:        if  $l(a) + \text{weight}(b) = l(b)$  then
24:           $G_L = G_L \cup \{a \rightarrow b\}$ 
25:        end if
26:      end if
27:    end for
28:  end while
29:  while  $B \neq \emptyset$  do
30:     $b = \text{extractMin}(B)$  // b for backwards
31:    for  $a \in \text{pred}(G, b)$  do
32:      if  $\text{level}(b) < \text{level}(a) + \text{weight}(a)$  then
33:         $\text{level}(b) = \text{level}(a) + \text{weight}(a)$ 
34:         $\text{insert}(R, b)$ 
35:         $\text{insert}(U, u)$ 
36:      end if
37:    end for
38:  end while
39: else
40:   if  $l(u) + \text{weight}(v) = l(v)$  then
41:      $G_L = G_L \cup \{u \rightarrow v\}$ 
42:   end if
43: end if
44: return  $U$ 

```

Algorithm 4 popVertex(G, G_{UL}, G_{WL}, R)

```

1: Input: Digraph  $G$ , weighted and unweighted auxiliary structures  $G_{UL}, G_{WL}$ , priority
   queue  $R$ 
2: Effect: Structures  $G, G_{UL}, G_{WL}$ , and  $R$  are updated
3:  $v = \text{top}(R)$ 
4:  $U = \text{popVertexUpdateDownstream}(G_{UL}, v)$ 
5:  $U = U \cup \text{popVertexUpdateDownstream}(G_{WL}, v)$ 
6:  $S = \text{succ}(G_{UL}, v)$ 
7: while  $S \neq \emptyset$  do
8:    $s = \text{top}(S)$  //  $s$  for successor
9:    $R.\text{push}(s)$ 
10: end while
11: while  $U \neq \emptyset$  do
12:    $u = \text{top}(U)$  //  $u$  for updated
13:    $R.\text{update}(u)$ 
14: end while
15:  $G = G \setminus \{u \rightarrow v\}$ 

```

Algorithm 5 computeAffected(G, G_L, w)

```

1: Input: Digraph  $G = (V, E)$ , structure  $G_L$ , vertex  $w$  to be removed
2: Output: List  $A$  of affected vertices
3:  $Q = \{w\}$ 
4:  $A = \emptyset$ 
5: while  $Q \neq \emptyset$  do
6:    $u = \text{dequeue}(Q)$ 
7:    $A = A \cup \{u\}$ 
8:   for  $v \in \text{succ}(G_L, u)$  do
9:      $G_L = G_L \setminus \{u \rightarrow v\}$ 
10:    if  $\text{pred}(G_L, v) = \emptyset$  then
11:       $\text{insert}(Q, v)$ 
12:    end if
13:  end for
14: end while
15: return  $A$ 

```

Algorithm 6 popVertexUpdateDownstream($G_L, u \rightarrow v$)

```

1: Input: Digraph  $G = (V, E)$ , structure  $G_L$ , edge  $u \rightarrow v$  to be removed
2: Output: List of updated vertices  $U$ 
3: Effect: Structure  $G_{UL}$  is updated
4:  $U = \emptyset$ 
5: if  $u \rightarrow v \in G_L$  then
6:    $G_L = G_L \setminus \{u \rightarrow v\}$ 
7:   if  $pred(G_L, v) = \emptyset$  then
8:      $A = \text{computeAffected}(G_L, v)$  //  $a$  for affected
9:     for  $a \in A$  do
10:       $\text{indeg}(a) = |pred(G, a) \cap A|$ 
11:    end for
12:     $Q = \text{queue}(\text{WeightedLongestFirst})()$ 
13:     $Q = \{a \in A \mid \text{indeg}(a) = 0\}$ 
14:    while  $Q \neq \emptyset$  do
15:       $q = \text{dequeue}(Q)$ 
16:       $l(q) = \max_{p \in pred(G, q)} l(p) + \text{weight}(q)$ 
17:       $G_L = G_L \cup \{p \rightarrow q \mid p \in pred(G, q) \wedge l(p) + \text{weight}(q) = l(q)\}$ 
18:      for  $a \in succ(G, q) \cap A$  do
19:         $\text{indeg}(a) = \text{indeg}(a) - 1$ 
20:        if  $\text{indeg}(a) = 0$  then
21:           $\text{insert}(Q, a)$ 
22:        end if
23:      end for
24:    end while
25:  end if
26: end if
27: return  $U$ 

```

Algorithm 7 merge($G, G_{UL}, G_{WL}, R_1, H, H_{UL}, H_{WL}, R_2, C$)

```

1: Input: Digraphs  $G, H$ , weighted and unweighted auxiliary structures
    $G_{UL}, G_{WL}, H_{UL}, H_{WL}$ , priority queues  $R_1, R_2$  related to graphs  $G, H$ , respectively, set
    $C$  of edges  $g \rightarrow h$  from vertex-set of  $G$  to vertex-set of  $H$ 
2: Effect: Structures  $G, G_{UL}, G_{WL}, R_1$  are updated
3:  $G = G \cup H, G_{UL} = G_{UL} \cup H_{UL}, G_{WL} = G_{WL} \cup H_{WL}$ 
4:  $R_1 = R_1 \cup (R_2 \setminus \{h \mid (g \rightarrow h) \in C\})$ 
5:  $Q = \text{queue}(\text{SumWeightedLongestFirst})(C)$ 
6: while  $Q \neq \emptyset$  do
7:    $(g \rightarrow h) = \text{dequeue}(Q)$ 
8:    $\text{insertEdge}(G, G_{UL}, G_{WL}, R, g \rightarrow h)$ 
9: end while

```

Using Differential Evolution to Identify Fuzzy Measures for the Exam Timetabling Problem

Edmund K Burke · Tiago Cardal Pais

Abstract In this work, we use the Differential Evolution algorithm to identify fuzzy measures for the Exam Timetabling Problem. For that purpose, we used some of previous results that can be found in the literature to help model the problem. We also described how to combine the Differential Evolution algorithm with the construction algorithm to tune the fuzzy measures. This method is tested on some benchmark tests. Moreover, it was shown that it can effectively improve some of the previous results based on the same construction heuristic. Furthermore, it also obtained some of the best results in the literature when compared with the best construction heuristics or construction hyper-heuristics found in the literature.

Keywords Differential Evolution, Exam Timetabling, Fuzzy Measure, Choquet Integral, Construction Heuristics

1 Introduction

Problems related to timetabling are present in daily life. Solving timetabling problems is a crucial task and affects many institutions and services such as hospitals, transportation enterprises, educational establishments, amongst many others. These problems have been the subject of increasing interest for the research community. Many high impact challenges are being addressed, particularly in the field of Operations Research and Artificial Intelligence, to solve timetabling problems in sports [12, 30], transportations (bus,railways,planes) [16, 7, 23], schools [1, 11, 26, 15, 28] and universities (a recent survey in this topic is presented by Qu et al. [25])

A general definition of timetabling can be seen in [6]:

“A timetabling problem is a problem with four parameters, T a finite set of times, R a finite set of resources, M , a finite set of meetings: and C , a finite set

Edmund K Burke
ASAP group, University Of Nottingham, Nottingham, NG8 1BB, UK
E-mail: ekb@cs.nott.ac.uk

Tiago Cardal Pais
ASAP group, University Of Nottingham, Nottingham, NG8 1BB, UK
E-mail: tpp@cs.nott.ac.uk

of constraints. The problem is to assign times and resources to the meetings so as to satisfy the constraints as far as possible.”

In this paper we are addressing the exam timetabling problem.

The state-of-the-art of construction hyper-heuristics and construction algorithms is summarized below. The results presented in these papers are compared to the results of the algorithm proposed in this paper in Section 7.

In Burke and Newall [4], an iterated construction algorithm is described. They make use of a construction ordering heuristic as the basic method for scheduling the exams. However, the authors introduce an iterated adaptive method which consists of changing the “degree” of each exam in each iteration. They proposed an incremental and exponential adaptation scheme. In the first case, the “degree” is modified by one unit at each iteration. On the other hand, the exponential scheme increments the “degree” by 2^n , where n is the number of iterations by which a particular “degree” was modified. Moreover, they compare the performance of the algorithm when using different basic ordering heuristics. They use the largest degree first (*LD*), a flat ordering (which initializes every “degree” on 0), smallest degree first (*SmD*), saturation degree (*SD*) and random ordering. They tested the algorithm using the Toronto data set [9]. It can be observed that the adaptation mechanism helps to improve the initial timetable given by the original order of the exams.

In Asmuni et al. [2], the authors used a fuzzy multiple heuristic ordering approach where a simple heuristic ordering was implemented, which was based on the Carter and Laporte [8] algorithm. The following three different criteria were used to order exams: (1) largest degree (*LD*); (2) largest enrolment (*LE*); (3) and least saturation (*SD*) degree criterion. They use a fuzzy inference system to combine the different criteria previously mentioned. All possible combinations were tested (*LD + LE*, *SD + LE* and *LD + SD*). The Mandani type fuzzy system that they used has a 9 rule structure, meaning that two linguistic variables were employed to evaluate the exam “quality” and, for each variable, three linguistic terms were defined: “small”, “medium” or “high”. The output linguistic variable “examweight” is also defined by the same three linguistic terms. A pre-normalisation of data was also performed before computing the fuzzy inference system. In this process, a linear transformation was used. Furthermore, a tuning process was also implemented. It consists, basically, of changing simultaneously, by small steps, the upper bound, centre and lower bound of the three membership functions. All results obtained for each instance used the best “tuned system”. All approaches were tested using the Toronto data set [9]. They conclude that the approach using a tuned fuzzy system with the *SD* and *LE* as input variables gave, overall, the best results.

Qu et al. [24] presented an adaptive hybridisation of basic graph heuristics within a graph hyper-heuristic framework. They first started studying some statistical properties of a random constructive graph hyper-heuristic. They observed that sequences of *SD* hybridised with Largest Weighted Degree (*LWD*) gave better results than if hybridised with *LD* or *LE*. Following that, they proposed an adaptive approach which uses sequences of *SD* hybridized with *LWD*. This new method consists of two steps. Firstly, they iteratively hybridise the *LWD* heuristic into the first half of a sequence based on *SD*. At the end of each iteration, if the solution obtained was feasible, the hybridisation amount was increased by 0.03. On the other hand, if that solution was feasible but had a higher cost than the best solution previously found, then the hybridisation amount was decreased by 0.01. Secondly, they hybridised the entire sequence

with *LWD*. However, in this step only the best sequences obtained from the previous step were used.

The main focus of this paper is to explain how one can use differential evolution to identify fuzzy measures which are then indirectly used to generate exam timetables. The timetables are built using the construction heuristic presented in [21]. Construction heuristics are useful since they are a tool which can quickly generate good exam timetables. They can also be combined with local search methods. Firstly, it would very quickly provide a good initial solution for the local search method; hence, the local search method could focus on improving the solution instead of wandering around trying to find an initial feasible solution. Moreover, if we decide to use a population based method, it would rapidly initialise the entire population. There are several different construction heuristics for the timetable problem [9]. These are also very closely related to the graph colouring problem since both problems can be considered to have the same basic structure [8]. However, Pais and Burke [21] showed that by combining some of these heuristic values with a Choquet integral, the resulting timetables were better than the individual construction heuristics, at approximately the same computational cost.

The paper is organized as follows. In Section 2, a mathematical formulation for the Exam Timetabling problem is presented. A brief introduction to very important concepts for this work is given in Section 3. This is followed by a description of the construction heuristic in Section 4. In Section 5, we show how to combine the Differential Evolution method with the construction algorithm to tune the fuzzy measures. A description of the experimental design is given in Section 6 followed by the computational results and discussion in Section 7. We finish this work with some conclusions presented in Section 8.

2 The Exam Timetabling Problem

The exam timetabling problem can be formulated as a combinatorial optimisation problem. In order to compare our approach with other methods proposed in the literature, we adopt the following formulation followed by many authors [25]. Let,

$$E = \text{total number of exams,} \quad (1)$$

$$P = \text{total number of periods,} \quad (2)$$

$$S = \text{total number of students,} \quad (3)$$

$$c_{ij} = \text{number of students enrolled in exam } i \text{ and } j \text{ for } i, j = 1, \dots, E, \quad (4)$$

$$a_{ij} = \begin{cases} 1 & \text{if } c_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \text{ for } i, j = 1, \dots, E, \quad (5)$$

and consider the following variables:

$$x_i = \text{period in which exam } i \text{ is scheduled, } i = 1, \dots, E \quad (6)$$

The formulation is:

$$\text{minimize } z = \frac{\sum_{i=1}^E \sum_{j=1}^E \text{proximity\_cost}(x_i, x_j) * c_{ij}}{2S} \quad (7)$$

$$\text{subject to: } |x_i - x_j| \geq a_{ij} \text{ for } i, j = 1, \dots, E \text{ and } i \neq j \quad (8)$$

$$1 \leq x_i \leq P \text{ and integer for } i, j = 1, \dots, E \quad (9)$$

where proximity cost is defined as:

$$\text{proximity\_cost}(x_i, x_j) = \begin{cases} 2^{5-|x_i-x_j|} & \text{if } 0 < |x_i - x_j| < 6 \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

for $i, j = 1, \dots, E$ and $i \neq j$

The objective function (7) penalizes the proximity of exams with students in common, using as weights the number of students involved in both examinations and a factor that depends on the proximity of the periods, ranging from 16 to 0 [9]. Constraint (8) ensures that any two exams indexed by i and j with students in common are not assigned to the same period.

3 Essential Concepts

In this section, a description of essential concepts for our work is provided. The notation presented here is used throughout the paper.

3.1 Fuzzy Sets

Definition 1 (Zimmermann [34]) *If $\mathcal{X}$ is a collection of objects designated by x then a fuzzy set $\tilde{A}$ in X is defined by a set of pairs:*

$$\tilde{A} = \{(x, \mu(x)) | x \in X\}$$

where $\mu_{\tilde{A}}(x)$ is the membership function of x in $\tilde{A}$.

From now on we are going to refer to a membership function as $f_{\tilde{A}}$ instead of the traditional way, as presented in the previous definition, $\mu_{\tilde{A}}$, to avoid any confusion with the representation of a fuzzy measure μ (see Section 3.3).

Fuzzy sets are often represented by triangular, trapezoidal (triangular as a particular case) or Gaussian membership functions. In general, a trapezoidal function is given by the following membership function:

$$f_{\tilde{A}} : D \subset X \rightarrow [0, 1]$$

Note that

$$f_{\tilde{A}}(x) = \begin{cases} 0 & , \text{ if } x \leq a \vee x > d \\ \frac{x-a}{b-a} & , \text{ if } a < x \leq b \\ 1 & , \text{ if } b < x \leq c \\ \frac{d-x}{d-c} & , \text{ if } c < x \leq d \end{cases} \quad (11)$$

where $x \in D \subset \mathcal{X}$ and $\tilde{A}$ is a fuzzy set in $\mathcal{X}$.

The triangular function is a trapezoidal function where $b = c$.

3.2 Linguistic Variable

The linguistic values or terms of a linguistic variable are concepts defined by words or expressions of a natural language.

Definition 2 (Zadeh [33]) A linguistic variable is characterized by the quintuple $(\mathcal{H}, T(\mathcal{H}), U, G, M)$, where $\mathcal{H}$ is the name of the variable, $T(\mathcal{H})$ is the set of terms or linguistic values of $\mathcal{H}$, U is the universe of the variable, G the semantic rule that generates the terms in $T(\mathcal{H})$ and M is the semantic rule associating each term or linguist value to its meaning trough the fuzzy set $M(X)$ ($M(X)$ is a fuzzy set on U).

3.3 Fuzzy Measures

Since here we are working with finite spaces, we are going to present a simplified definition of a fuzzy measure. More details about this topic can be found in Wang and Klir [32].

Definition 3 (Murofushi and Sugeno [20]) A fuzzy measure, also called non-additive measure, μ , defined on the measurable space $(X, \mathfrak{X})$ is a set function $\mu : \mathfrak{X} \rightarrow [0, +\infty[$ satisfying the following axioms:

- i $\mu(\emptyset) = 0$
- ii $A \subseteq B \Rightarrow \mu(A) \leq \mu(B)$ (monotonicity).

where $X = \{x_1, \dots, x_n\}$, and $\mathfrak{X}$ is a σ -algebra over the space X . $(X, \mathfrak{X}, \mu)$ is said to be a fuzzy measure space.

Remark 1 In general, $\mu(X)$ can be any positive finite (or infinity) quantity. Nevertheless, a fuzzy measure that verifies $\mu(X) = 1$ in addition to the properties described in Definition 3 is said to be a *regular* fuzzy measure [17].

Proposition 1 (Burke and Pais [5]) Consider a finite set $X = \{x_1, \dots, x_n\}$ and the measurable space $(X, \mathfrak{X})$, where $\mathfrak{X}$ is a σ -algebra over the set X . Consider also a set function $\zeta : \mathfrak{X} \rightarrow \mathbb{R}_0^+$ defined as:

$$\zeta(A) = \begin{cases} 0 & , \text{ if } A = \emptyset \\ \max_{\substack{B \in \mathfrak{X} \\ B \subset A}} \{\zeta(B)\} + d_A & , \text{ otherwise} \end{cases} \quad (12)$$

where $d_A \in \mathbb{R}_0^+$.

Then the ζ set function is a fuzzy measure.

The fuzzy measure of A can be regarded as the overall importance of criteria A . In a similar way, the d_A values, in Function (12), represent at least how much more important A is over all its subsets. For instance, consider $d_{\{x_1\}} = 0.1$ and $\mu(x_1) = 0.7$, where $X = \{x_1, x_2\}$. This can be perceived as the criterion $\{x_1\}$ being more important than $\emptyset$ by 0.1 units. However, the overall importance of criterion $\{x_1\}$ is 0.7.

We will define ζ_0 to be the set function where the selected σ -algebra is the same as the power set of X ($\mathcal{P}(X)$). As we will see later, this set function was extremely useful when identifying a fuzzy measure using the DE algorithm.

If we assign the same value to all the d_A , $A \in \mathcal{P}(X)$, the outcome fuzzy measure will be in the equilibrium state. Hence, the Choquet integral will be equivalent to a simple average. Moreover, to build an additive measure we just need to define the d_A values according to Equation 13. In this case, the Choquet integral coincides with Lebesgue integral, which is the same as a weighed sum. Furthermore, to build a super-additive (sub-additive) measure the d_A values need to be greater (less) than the values given by Equation 13.

$$d_A = \sum_{I \in J} \mu_I \quad (13)$$

where $A \in \mathcal{P}(X) - \{\emptyset\}$;

$$J = \{B \in \mathcal{P}(X) : B \subset A \wedge |B| = |A| - 1 \wedge \mu(B) \neq \max_{\substack{K \in \mathcal{P}(X) \\ K \subset A \\ |K|=|A|-1}} \{\mu(K)\}\}$$

3.4 Choquet Integral

Again, as in Section 3.3, we present a definition of the Choquet integral [10] for the particular case of discrete spaces.

Definition 4 (Grabisch [13]) *Let $(X, \mathcal{P}(X), \mu)$ be a fuzzy measure space. The Choquet integral of a function $f : X \rightarrow [0, 1]$ with respect to μ is defined by*

$$\mathfrak{C}_\mu(f_1, \dots, f_n) = \sum_{i=1}^n (f_{(i)} - f_{(i-1)}) \mu(A_{(i)})$$

where $\mathcal{P}(X)$ is the power set of X , $f_i := f(x_i)$, $i = 1, \dots, N$, $\cdot_{(i)}$ indicates a permutation such as $0 \leq f_{(1)} \leq \dots \leq f_{(n)} \leq 1$, $A_{(i)} = \{x_{(i)}, \dots, x_{(n)}\}$, and $f_{(0)} = 0$.

The Choquet integral is described in more detail in [20].

3.5 Differential Evolution

The differential evolution (DE) algorithm, proposed by Price and Storn [29, 22], is a successful continuous optimization method. Moreover, DE is a simple and very powerful algorithm for global and continuous optimization problems. It has been successfully applied to parameter tuning [31, 27, 22, 3, 18].

The classical DE algorithm is a stochastic population-based method. This method has three control parameters that are constant during the entire search. These parameters are: amplification factor - F ; crossover probability - CP ; and population size -

PS. The population consists of D -dimension vectors ($\mathbf{x}_{iG} = (x_{1iG}, \dots, x_{DiG})$, where $x_{jiG} \in [x_j^{lb}, x_j^{ub}]$, G is the generation number and $i = 1, \dots, PS$). The algorithm starts by initialising the population $\mathbf{x}_{i1} = (x_{1i1}, \dots, x_{Di1})$, where $i = 1, \dots, PS$, by uniformly drawing a random number between the lower bound and the upper bound.

Like most evolutionary algorithms, DE uses mutation and crossover operators to evolve the population from one generation to the next. Both operators are used to produce a trial vector - $\mathbf{u}_{iG}$ - for each target vector - $\mathbf{x}_{iG}$. We use the selection operator, presented in Equation 14, to select the best individual for the new population. The pseudo-code of the algorithm is depicted in Algorithm 1.

$$\mathbf{x}_{i(G+1)} = \begin{cases} \mathbf{u}_{iG} & , \text{if } z(\mathbf{u}_{iG}) < z(\mathbf{x}_{iG}) \\ \mathbf{x}_{iG} & , \text{otherwise} \end{cases} \quad (14)$$

where $z(\cdot)$ is the fitness function.

Algorithm 1 DE algorithm pseudo-code

```

initialise the new population
for  $G = 1$  to  $MAXGEN$  do
  for  $i = 1$  to  $PS$  do
    create a mutant vector ( $\mathbf{v}_{iG}$ ) using a mutation operator
    create a trial vector ( $\mathbf{u}_{iG}$ ) using crossover operator between  $\mathbf{v}_{iG}$  and  $\mathbf{x}_{iG}$ 
    select the new individual ( $\mathbf{x}_{i(G+1)}$ ) using the selection operator between  $\mathbf{u}_{iG}$  and  $\mathbf{x}_{iG}$ 
  end for
  update the population with the new individuals
end for

```

where the selection operator is defined by Equation 14, the mutation operator is defined in Section 3.5.1 and the crossover operator is defined in Section 3.5.2.

3.5.1 Mutation Operator

For every candidate vector ($\mathbf{x}_{iG}, i = 1, \dots, PS$), a mutation operator is used to create a new mutated vector ($\mathbf{v}_{iG}$). Here, we describe four of the original mutation operators. Moreover, the same notation that used in [29] is adopted.

$$\text{"DE/best/1"} \quad \mathbf{v}_{iG} = \mathbf{x}_{\text{best}G} + F \times (\mathbf{x}_{r_1G} - \mathbf{x}_{r_2G})$$

$$\text{"DE/rand/1"} \quad \mathbf{v}_{iG} = \mathbf{x}_{r_1G} + F \times (\mathbf{x}_{r_2G} - \mathbf{x}_{r_3G})$$

$$\text{"DE/best/2"} \quad \mathbf{v}_{iG} = \mathbf{x}_{\text{best}G} + F \times (\mathbf{x}_{r_1G} - \mathbf{x}_{r_2G} + \mathbf{x}_{r_3G} - \mathbf{x}_{r_4G})$$

$$\text{"DE/rand/2"} \quad \mathbf{v}_{iG} = \mathbf{x}_{r_1G} + F \times (\mathbf{x}_{r_2G} - \mathbf{x}_{r_3G} + \mathbf{x}_{r_4G} - \mathbf{x}_{r_5G})$$

where $\mathbf{x}_{\text{best}G}$ is the best individual in the population at generation G ; r_1, r_2, r_3, r_4 and r_5 are mutually different random integer numbers between 1 and PS . Moreover, the latter indexes should also be different from index i .

3.5.2 Crossover Operator

After obtaining the mutated vector ($\mathbf{v}_{iG}$), we use a “binomial” crossover operator to obtain a trial vector ($\mathbf{u}_{iG}$). The crossover is done between the mutated vector ($\mathbf{v}_{iG}$) and the candidate vector ($\mathbf{x}_{iG}$). The operator is defined below by Equation (15).

$$u_{jiG} = \begin{cases} v_{jiG} & , \text{if } urn_j \leq CP \text{ or } j = k \\ x_{jiG} & , \text{if } urn_j > CP \text{ and } j \neq k \end{cases} \quad (15)$$

$$\forall j \in \{1, \dots, D\}$$

where urn_j is a uniform random number drawn from unit interval; and k is an integer random number drawn from $\{1, \dots, D\}$.

There is another type of crossover called “exponential” crossover. However, this type of crossover is rarely used in practice [3].

4 The Construction Heuristic

We implement the same construction heuristic as described in Pais and Burke [21]. The construction heuristic block is composed by two secondary order heuristics. These are: the exam ordering heuristic; and the period ordering heuristic. The former concerns the order in which exams are scheduled. The latter chooses the period in which a particular exam should be scheduled. Furthermore, we also implemented a backtracking procedure which is based on the one described in [9]. A pseudo-code of the algorithm is depicted in Algorithm 2.

Algorithm 2 Construction algorithm pseudo-code

```

unscheduled_exams  $\leftarrow$  all exams
while unscheduled_exams is not empty do
    Select an exam from the list unscheduled_exams using a exam_ordering_heuristic
    Select a feasible period to schedule the previous selected exam using a
    period_ordering_heuristic
    if it is impossible to select a feasible period then
        Use the backtracking procedure described in [9]
        if it is impossible to backtrack then
            break
        end if
    end if
    Update unscheduled_exams list
end while

```

In this work, we use a *deterministic&random* ordering for both exams and periods. For the exam ordering, this method consists of sorting the exams according to the approach described in Section 4.1; ties are broken randomly. For the period ordering, the method is similar. The periods are sorted according to the weights obtained by using Eq. 7 and ties are broken randomly.

4.1 A Construction Heuristic Using The Choquet Integral

The more traditional aggregation methods (e.g. Weighted Sum, Ordered Weighted Average) are easy to interpret but are too restrictive since they are not able to represent the interaction between the criteria. It is assumed that the criteria are independent, when in most practical cases this does not happen. This method is motivated by the fact that we can represent the information given by the basic heuristics values (given below), as well as the interaction between those heuristics values by using a fuzzy measure (see Section 3.3). In this way we can model criteria that are not independent.

The basic heuristic values represent how “hard” it is to schedule a particular exam. Each of the five basic heuristics are described below.

1. Saturation Degree (SD): the number of feasible periods in which an exam can be scheduled;
2. Colour Degree (CD): the number of total conflicts that an exam has with the already scheduled exams;
3. Largest Degree (LD): the number of total conflicts;
4. Largest Weighted Degree (LWD): the number of total conflicts weighted by the number of students involved in each one;
5. Largest Enrolment (LE): the number of enrolments;

The Choquet integral can be regarded as an extension of the Lebesgue integral. That is, if the measure at hand is additive, or in other words, is a classical measure, then the Choquet integral coincides with the Lebesgue integral [20]. Therefore, it is easier to interpret its output than other fuzzy integrals. Moreover, the Choquet integral has some algebraic properties that other fuzzy integrals do not have [13]. Hence, it makes it more suitable for multicriteria decision making problems [20, 13, 14].

Since the information given by the basic heuristic values had different units, we used fuzzy sets (see Section 3.1) to normalise the values into the unit interval. The goal of using this information is to decide which exam is “harder” to schedule. Bearing this in mind, we modelled each basic heuristic as a linguistic variable (see Section 3.2). Hence, we used a triangular membership function (see Eq. 11) to represent the linguistic terms, such as “low SD” value, “high CD” value, “high LD” value, “high LWD” value and “high LE” value. The membership functions are relative to each iteration, e.g., if the highest SD value is 5 (in one particular iteration) then all exams with that value are going to have a membership of 0. The membership functions for each linguistic term are defined below.

$$f_{\widetilde{lowSD}}(x_{SD}) = \frac{maxSD - x_{SD}}{maxSD} \quad (16)$$

$$f_{\widetilde{r}}(x_s) = \frac{x_s}{t} \quad (17)$$

where $r \in \{highCD, highLD, highLWD, highLE\}$; $maxSD$ is the maximum SD value in the current iteration. Also $t \in \{maxCD, maxLD, maxLWD, maxLE\}$ is the maximum value for $s \in \{CD, LD, LWD, LE\}$ in the current iteration, respectively.

Moreover, if some of the $maxCD, maxLD, maxLWD, maxLE$ values are equal to zero, in some iteration, then the respective function returns 0 by default. On the other hand, if $maxSD$ value is equal to zero then the function returns 1 by default.

After all the values are fuzzyfied and a fuzzy measure is set, we can use the Choquet Integral (see Section 3.4) to combine all the information. As we did with the other basic heuristics, the exams are ordered decreasingly according to the values obtained by using this method. That is, if one exam is attributed value 1 then this means that it is very hard to schedule, according to the information given by the five basic heuristic values. Hence, it should be scheduled before all other exams.

To better illustrate how the process works, an example is presented here. Consider two exams, e_1 and e_2 . After the basic heuristic values were computed and normalized, we obtained the following values: $x_{e_1} = (0.4, 0.5, 0.5, 0.6, 0.8)$ and $x_{e_2} = (0.8, 0.4, 0.7, 0.7, 0.4)$, corresponding to the SD , CD , LD , LWD , LE heuristics, respectively. The Choquet integral value for exam e_1 is computed as:

$$\begin{aligned}\mathfrak{C}_\mu(x_{e_1}) = & \mu_{\{SD, CD, LD, LWD, LE\}} x_{e_1 SD} + \\ & + \mu_{\{CD, LD, LWD, LE\}} (x_{e_1 CD} - x_{e_1 SD}) + \\ & + \mu_{\{LD, LWD, LE\}} (x_{e_1 LD} - x_{e_1 CD}) + \\ & + \mu_{\{LWD, LE\}} (x_{e_1 LWD} - x_{e_1 LD}) + \\ & + \mu_{\{LE\}} (x_{e_1 LE} - x_{e_1 LWD})\end{aligned}$$

An analogous process is also applied to exam e_2 . The next exam to be scheduled would be the one that obtains the highest value.

5 Identifying Fuzzy Measures for the Exam Timetabling Problem Using Differential Evolution

In this section, we show how to combine the set function ζ_0 (see Equation 18) with DE (see Section 3.5) to identify a fuzzy measure for the exam timetabling problem, which is described in Section 2.

$$\zeta_0(A) = \begin{cases} 0 & , \text{ if } A = \emptyset \\ \max_{\substack{B \in \mathcal{P}(X) \\ B \subset A \\ |B|=|A|-1}} \{\zeta(B)\} + d_A & , \text{ otherwise} \end{cases} \quad (18)$$

where $d_A = v_A^2$ and $v_A \in \mathbb{R}$.

The Function (18) gives us a fuzzy measure. This fuzzy measure represents the importance or, in this case, how "hard" it would be to schedule an exam when criteria $A \in \mathcal{P}(X)$, $X = \{SD, CD, LD, LWD, LE\}$, are fulfilled. Hence, as explained in Section 3.3, the d_A values represent at least how much more difficult it is to schedule an exam if criteria A are fulfilled than to schedule an exam if criteria $B \subset A, |B| = |A| - 1$, are fulfilled. For instance, consider $\mu(\{SD, CD\}) = 0.3$ and $d_{\{SD, CD\}} = 0.4$. In a conceptual way, this means that an exam is 0.3 units "hard" to schedule if criteria $\{SD, CD\}$ is fulfilled. On the other hand, it is at least 0.4 units more difficult to schedule an exam where $\{SD, CD\}$ are fulfilled than to schedule an exam where $\{SD\}$ or $\{CD\}$ are fulfilled.

In this work, each individual vector of the population indirectly represents a fuzzy measure. This means that a vector has the necessary information to build a fuzzy measure. In this case, vectors contain the distances - $v_A \in \mathbb{R}, \forall A \in \mathcal{P}(X) - \emptyset$. These distance values are then used by the ζ_0 function. The DE population is initialised by loading each vector with uniform random values drawn from the unit interval.

For example, consider the vector $\mathbf{x} = (0.1, 0.2, 0.3)$. In this case, the set X only has two elements; hence $\mathcal{P}(X) = \{\emptyset, \{X_1\}, \{X_2\}, \{X_1, X_2\}\}$. Therefore, the fuzzy measure associated with $\mathbf{x}$ is $\zeta_0(\emptyset) = 0$, $\zeta_0(\{X_1\}) = \zeta_0(\emptyset) + 0.1^2 = 0.01$, $\zeta_0(\{X_2\}) = 0.04$ and $\zeta_0(\{X_1, X_2\}) = \max\{0.01, 0.04\} + 0.3^2 = 0.13$. In this example, we do not have a regular fuzzy measure. However, this can be easily done by dividing all the components of the fuzzy measure by $\zeta_0(\{X_1, X_2\})$.

Since our objective is to find the fuzzy measure that gives the best timetables using the construction algorithm described in Section 4, the fitness function consists of an average obtained by 40 runs of the construction algorithm. As it has been described, each individual represents a fuzzy measure that is “extracted” using the set function ζ_0 . With this fuzzy measure we ran the algorithm 40 times. We compute the average of these 40 values and that represents the fitness value of one individual.

All the parameters used in this work were taken from [5] with the exception of the population size and number of generations. In this work it would be impractical to use a population size of 320 and 100 generations since the algorithm would take a very long time to run. Hence, we decided to use a population size of 160 and 50 generations since it also gave good results in [5]. The uniform crossover operator is used. For the mutation, we used the “DE/best/1” operator. The crossover probability and amplification factor are 0.99 and 0.4, respectively. We selected this configuration since they were the parameters which presented the best results in [5].

6 Experimental Design

To test the performance of the algorithm, we used two data sets. The first one was a collection of real problems and is available in an online repository created by Michael Carter¹, which from now on we refer to as Toronto data sets. The second data set was put online for the International Timetabling Competition² (ITC 2007) [19]. In this work, we run the DE for each of the instances using the parameters defined in Section 5. Since the fitness function has a random component we decided to select the best 10% of the population, i.e. we selected the best 16 fuzzy measures of the last population. We used the average performance in order to compare the success of the 16 fuzzy measures for a particular data set. For this purpose, we performed 3300 runs for each fuzzy measure and for each data set. If any two measures for the same data set presented the same average value (rounded to 2 decimal units), then the one with the lowest minimum value (rounded to 2 decimal units) was selected. In the case of a tie, we selected the measure with the lowest maximum value (rounded to 2 decimal units).

We ran the construction algorithm 3300 times using the fuzzy measure presented in [21].

We present some descriptive statistics (see Tables ?? and 1) of the 3300 samples for each instance and for the “best” fuzzy measure and the fuzzy measure presented in

¹ <ftp://ftp.mie.utoronto.ca/pub/carter/testprob>

² <http://www.cs.qub.ac.uk/itc2007/>

[21]. Moreover, for each data set and for both fuzzy measures, we computed averages of 50 samples over the 3300. In this way, we have obtained 66 samples of the average performance for each data set for both fuzzy measures. We performed a statistical test (the Sapiro Wilks normality test, with significance level of 0.95) to check if the samples of the averaged values followed a normal distribution. The normality distribution hypothesis was rejected for the ITC4 instance. To compare the performance of DE “best” measure with the measure presented in [21], we used the statistical t-test with a significance level of 0.95. The following null hypothesis was used:

$$H0 : \mu_{df_i DE-CI} \geq \mu_{df_i CI} \quad (19)$$

where μ represents the mean value and df_i is the data file $i \in \{\text{Toronto data files, ITC's data files}\}$. $DE - CI$ represent the best fuzzy measures found by DE; whereas CI represent the fuzzy measure presented in [21]. However, we did not apply the t-test for ITC5 and ITC9 instances since they failed to pass the normality test.

In the next section, we also compare the minimum value given by the DE “best” fuzzy measure with the state of the art results.

7 Computational Results

Table 1 depict the minimum (min), maximum (max) and average (avg) values over the 3300 runs for all data sets (columns) and for the DE “best” measure as well as the measure provided in [21] (lines). The best results are presented in boldface.

Table 2 represents the p-Values of the t-test for each instance. The instances that do not have any value correspond to ITC5 and ITC9 data since they did not follow the prerequisites for the t-test.

Table 3 depicts the best results in the literature for the Toronto benchmark. It also contains the best results of some constructive methods or constructive hyper-heuristic methods. The best results are displayed in boldface. The best results of the aforementioned methods are followed by a check sign.

7.1 Discussion

We can observe from Table 1 that, in most instances, the “best” fuzzy measure identified by DE performs better than the fuzzy measure from a previous study [21]. This method was able to improve the average results of the construction heuristic on 24 (of 25) data sets. Moreover, these improvements were statistically significant in 21 instances, as can be observed in Table 2. For the instances that the DE algorithm was not able to improve the previous results, we performed some more statistical tests. On hec-s-92 and yor-f-83 data sets, we do not reject the hypothesis that the DE algorithm performs statistically equal to the previous fuzzy measure. Nevertheless, DE statistically performs better in 91% of the cases where a t-test could be used. Furthermore, these results were improved by 4% on average. The best improvement was in ITC12 by 12.9%; whereas the lowest improvement was in tre-s-92 by 0.3%. It can also be observed that the worst performance was in yor-f-83 where the result was worse by a factor of 0.0023.

The worst performance was registered for the maximum values where only the results of 23 data sets were better than the previous results. The average performance

Table 1: Computational results for the Toronto and ITC2007 data sets

Data Set		car-f-92	car-s-91	ear-f-83	hec-s-92	kfu-s-93
CI	min	4.42	5.16	36.97	12.01	15.08
	max	7.13	6.65	60.51	23.53	23.61
	avg	5.02	5.59	44.21	15.85	17.47
DE-CI	min	4.31	5.01	36.85	12.24	14.38
	max	5.97	6.14	58.42	23.04	22.46
	avg	4.76	5.38	43.37	15.77	16.72
Data Set		lse-f-91	pur-s-93	rye-s-93	sta-f-83	tre-s-92
CI	min	11.35	4.87	10.26	162.79	8.62
	max	15.27	5.67	17.7	177.87	11.48
	avg	12.8	5.14	11.67	171.59	9.34
DE-CI	min	10.99	4.75	9.81	160.46	8.55
	max	14.38	5.72	15.86	177.43	11.21
	avg	12.49	5	11.33	168.44	9.32
Data Set		uta-s-92	ute-s-92	yor-f-83	ITC1	ITC2
CI	min	3.42	29.86	42.16	1.14	0.26
	max	4.52	40.66	55.96	1.41	0.4
	avg	3.68	33.79	47.38	1.26	0.32
DE-CI	min	3.38	27.97	39.69	1.11	0.25
	max	4.52	33.42	58.26	1.41	0.36
	avg	3.65	29.76	47.49	1.25	0.31
Data Set		ITC3	ITC4	ITC5	ITC6	ITC7
CI	min	1.86	14.36	0.46	4.78	0.1
	max	2.23	21.56	0.68	5.93	0.17
	avg	2.03	16.51	0.57	5.35	0.13
DE-CI	min	1.76	14	0.41	4.43	0.09
	max	2.11	21.36	0.66	5.6	0.15
	avg	1.94	16.11	0.53	4.76	0.12
Data Set		ITC8	ITC9	ITC10	ITC11	ITC12
CI	min	0.15	6.24	10.99	4.53	7.6
	max	0.27	8.12	12.43	5.3	11.79
	avg	0.21	7.04	11.66	4.93	8.51
DE-CI	min	0.14	6.23	10.6	4.29	7.13
	max	0.24	7.66	12.28	5.22	7.89
	avg	0.19	6.95	11.53	4.71	7.41

Table 2: p-Values for all the data Sets

Data Set	car-f-92	car-s-91	ear-f-83	hec-s-92	kfu-s-93
p-value	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$	0.09537	$< 10^{-16}$
Data Set	lse-f-91	pur-s-93	rye-s-93	sta-f-83	tre-s-92
p-value	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-5}$
Data Set	uta-s-92	ute-s-92	yor-f-83	ITC1	ITC2
p-value	$< 10^{-16}$	$< 10^{-16}$	0.93615	$< 10^{-4}$	$< 10^{-16}$
Data Set	ITC3	ITC4	ITC5	ITC6	ITC7
p-value	$< 10^{-16}$	$< 10^{-16}$	-	$< 10^{-16}$	$< 10^{-16}$
Data Set	ITC8	ITC9	ITC10	ITC11	ITC12
p-value	$< 10^{-16}$	-	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$

decrease on the 2 data sets was around 2.5%. Notwithstanding, the average improvement for maximum values was around 7%.

Table 3: Best computational results of some constructive methods and the DE-CI for the Toronto data set

Data Set	[8]	[4]	[2]	[24]	Best re- ported [25]	DE-CI
car-f-92	6.2	4.32	4.54	4.32	3.93	4.31 ✓
car-s-91	7.1	4.97 ✓	5.29	5.11	4.5	5.01
ear-f-83	36.4	36.16	37.02	35.56 ✓	29.3	36.85
hec-s-92	10.8 ✓	11.61	11.78	11.62	9.2	12.24
kfu-s-93	14.0 ✓	15.05	15.80	15.18	13.0	14.38
lse-f-91	10.5 ✓	10.96	12.09	11.32	9.6	10.99
pur-s-93	3.9	-	-	-	-	4.75
rye-s-93	7.3 ✓	-	10.38	-	6.8	9.81
sta-f-83	161.5	161.91	160.42	158.88 ✓	134.9	160.46
tre-s-92	9.6	8.38 ✓	8.67	8.52	7.9	8.55
uta-s-92	3.5	3.36	3.57	3.21 ✓	3.14	3.38
ute-s-92	25.8 ✓	27.41	28.07	28.00	24.4	27.97
yor-f-83	41.7	40.77	39.80	40.71	36.2	39.69 ✓

We can observe that a good performance was also achieved for the minimum values. In this scenario, the algorithm was able to improve the performance in 24 data sets. The hec-s-92 data set was the only instance where the best result was not achieved. Nevertheless, this result was only worse by 1.9%. The minimum results were improved by 4.2% on average. The highest improvement was 10.8% and this was registered in the ITC5 data set.

In this study, we also compared the DE-CI results with the state of the art results. It is not our intention to directly compete with local search methods. It is known that these methods perform better than construction methods. However, constructive methods are a valuable tool to build good quality timetables within a very short time period and, of course, we used to initialise local search methods and population based algorithms. Hence, the results shown in Table 3 in boldface are merely for comparison. Nevertheless, it can be seen that our algorithm performed well across the 12 Toronto data sets. A check sign follows the best results for construction hyper-heuristics or construction heuristics in Table 3. As can be seen, in this scenario, DE-CI obtained two best results in the car-f-92 and yor-f-83 data sets.

8 Conclusion

In this work, we described a DE method to tune fuzzy measures for the Exam Timetabling problem. The ζ_0 function also plays an important role in this method. This function ensures that all fuzzy measures satisfy the monotonicity constraints, that is, we did need to introduce these constraints in the model.

It can be seen that, in most cases, using the same construction algorithm after tuning the fuzzy measures greatly improves the previous results [21]. However, for two of the instances, it did not improve the previous results. Moreover, this method also gave the best results in some benchmark problems when compared to the best results found in the literature for construction hyper-heuristics or other construction heuristics.

Acknowledgements This research has been supported by Fundação para a Ciência e Tecnologia (FCT) Portugal grant number SFRH/BD/43486/2008.

References

1. D. Abramson, M. Krishnamoorthy, and H. Dang. Simulated annealing cooling schedules for the school timetabling problem. *Asia Pacific Journal of Operational Research*, 16:1–22, 1999.
2. H. Asmuni, E. K. Burke, J. M. Garibaldi, B. McCollum, and A. J. Parkes. An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. *Computers and Operations Research*, 36(4):981–1001, 2009.
3. J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, and V. Zumer. High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction. In *IEEE Congress on Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence)*, pages 2032–2039, 2008.
4. E. K. Burke and J. P. Newall. Solving examination timetabling problems through adaptation of heuristic orderings. *Annals of Operations Research*, 129(1):107–134, 2004.
5. E. K. Burke and T. C. Pais. Using differential evolution to identify fuzzy measures. Technical Report NOTTCS-TR-SUB-1105251404-2799, University of Nottingham, May 2011.
6. E. K. Burke, J. Kingston, and D. de Werra. Applications to timetabling. *Handbook of Graph Theory*, pages 445–474, 2004.
7. A. Caprara, M. Fischetti, P. L. Guida, M. Monaci, G. Sacco, and P. Toth. Solution of real-world train timetabling problems. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, page 10, 2001.
8. M. W. Carter and G. Laporte. Recent developments in practical examination timetabling. In *Selected papers from the First International Conference on Practice and Theory of Automated Timetabling*, pages 3–21, London, UK, 1996. Springer-Verlag.
9. M. W. Carter, G. Laporte, and S. Y. Lee. Examination Timetabling: Algorithmic Strategies and Applications. *Journal of the Operational Research Society*, 47(3): 373–383, 1996.
10. G. Choquet. Theory of capacities. *Ann. Inst. Fourier*, 5(131-295):54, 1953.
11. A. Colorni, M. Dorigo, and V. Maniezzo. Metaheuristics for High School Timetabling. *Computational Optimization and Applications*, 9(3):275–298, 1998.
12. K. Easton, G. Nemhauser, and M. Trick. Sports scheduling. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 2004.
13. M. Grabisch. Fuzzy integral in multicriteria decision making. *Fuzzy Sets and Systems*, 69(3):279–298, 1995.
14. M. Grabisch. The application of fuzzy integrals in multicriteria decision making. *European journal of operational research*, 89(3):445–456, 1996.
15. M. P. Hansen and R. V. V. Vidal. Planning of high school examinations in Denmark. *European Journal of Operational Research*, 87(3):519–534, 1995.

16. M. T. Isaai and M. G. Singh. Hybrid applications of constraint satisfaction and meta-heuristics to railway timetabling: a comparative study. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 31(1):87–95, 2001.
17. G. J. Klir, Z. Wang, and D. Harmanec. Constructing fuzzy measures in expert systems. *Fuzzy sets and Systems*, 92(2):251–264, 1997.
18. T. Krink, S. Paterlini, and A. Resti. Using differential evolution to improve the accuracy of bank rating systems. *Computational Statistics & Data Analysis*, 52(1): 68–87, 2007.
19. B. McCollum, P. McMullam, E. K. Burke, A. J. Parkes, and R. Qu. The second international timetabling competition: Examination timetabling track. Technical Report QUB/IEEE/Tech/ITC2007/Exam/v4.0/17, Queen’s University Belfast, September 20 2007.
20. T. Murofushi and M. Sugeno. An interpretation of fuzzy measures and the Choquet integral as an integral with respect to a fuzzy measure. *Fuzzy sets and Systems*, 29(2):201–227, 1989.
21. T. C. Pais and E. K. Burke. Choquet integral for combining heuristic values for exam timetabling problem. In *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010)*, Aug 2010.
22. K. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Springer-Verlag New York, Inc., 2005.
23. X. Qi, J. Yang, and G. Yu. Scheduling problems in the airline industry. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 2004.
24. R. Qu, E. K. Burke, and B. McCollum. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research*, 198(2):392–404, 2009.
25. R. Qu, E. K. Burke, B. Mccollum, L. T. Merlot, and S. Y. Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1):89, 2009.
26. G. Ribeiro Filho and L. A. N. Lorena. A Constructive Evolutionary Approach to School Timetabling. *Applications of Evolutionary Computing - Lecture Notes in Computer Science*, 2037:130–139, 2001.
27. J. Ronkkonen, S. Kukkonen, and K. V. Price. Real-parameter optimization with differential evolution. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, 2005.
28. A. Schaerf. Local search techniques for large high school timetabling problems. *Systems, Man and Cybernetics, Part A, IEEE Transactions on*, 29(4):368–377, 1999.
29. R. Storn and K. Price. Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4): 341–359, 1997.
30. M.A. Trick. A Schedule-Then-Break Approach to Sports Timetabling. *Lecture Notes in Computer Science*, pages 242–253, 2001.
31. R. K. Ursem and P. Vadstrup. Parameter identification of induction motors using differential evolution. In *Evolutionary Computation, 2003. CEC’03. The 2003 Congress on*, volume 2, 2003.
32. Z. Wang and G. J. Klir. *Fuzzy measure theory*. Plenum Publishing Corporation, 1992.

33. L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning, Part I. *Information Sciences*, 8(3):199–249, 1975.
34. H. J. Zimmermann. *Fuzzy Set Theory and its applications*. Kluwer Academic Publishers, 1996.

A Decomposition Approach for Shipment Pickup and Delivery with Time-Windows Under Uncertainty

Lavanya Marla · Cynthia Barnhart

Abstract We consider a special class of large-scale, network-based, resource allocation problems under uncertainty, namely that of multi-commodity flows with time-windows under uncertainty. In this class, we focus on problems involving shipment pickup and delivery with time windows, with the goal of gaining insights applicable to the broader problem class. Our work examines methods of proactive planning, that is, robust plan generation to protect against future uncertainty. By *a priori* modeling uncertainties in data corresponding to service times, resource availability, supplies and demands, we generate solutions that are more *robust* operationally, that is, more likely to be executed or easier to repair when disrupted. We propose a novel modeling and solution framework involving a decomposition scheme that separates problems into a routing master problem and scheduling sub-problems; and iterates to find the optimal solution. Uncertainty is captured in part by the master problem and in part by the sub-problem. Finally, we present computational experiments on a scaled-down version of a shipment carrier's network and on a larger airline's network.

1 Introduction

In this paper, we consider the class of large-scale network-based problems including multi-commodity flow problems with time-windows (*MCFTW*), which are at the core of problems arising in transportation, communications and logistics. Such resource allocation problems with their large-scale nature and associated complexity, have been ideal candidates for the application of optimization techniques. However, conventional optimization techniques usually involve assumptions of deterministic inputs, leading to solutions that are easily disrupted when realized parameter values are different; and thus exhibit a lack of *robustness* and high costs of recovery or repair. Such solutions are rarely (if ever) executed, and certainly, never truly optimal. In this work, our objective

Lavanya Marla
iLab, Heinz College, Carnegie Mellon University
E-mail: lavanyamarla@cmu.edu

Cynthia Barnhart
Massachusetts Institute of Technology
E-mail: cbarnhar@mit.edu

is to build robust network resource allocation solutions that: (i) are less fragile to disruption, (ii) easier to repair if needed, and (iii) minimize the realized, not just planned, problem costs.

Uncertainty in MCFTW can occur in the form of stochasticity in the supplies and demands of commodities; available capacities of the network links; and travel and service times on the network. We denote the multi-commodity flow problem with time-windows under uncertainty as *MCFTW-UU*. The MCFTW is at the core of network design problems, and hence, in finding robust solutions to the MCFTW-UU, we expect to provide insights into the more complex problem of network design under uncertainty.

1.1 Problem Description

To illustrate and evaluate our approach, we consider a specific problem, namely the *Shipment Routing Problem with Time Windows Under Uncertainty (SRTW-UU)*. Under SRTW-UU, for each vehicle v in the set of vehicles V , we are given a set of vehicle routes defining a network of locations with time-independent travel times and capacities u_{ij} corresponding to vehicle capacities on the links, and service times at locations. Each shipment k in the set of shipments K with demand d_k needs to be routed over this network, from its origin $O(k)$ to its destination $D(k)$. Transshipment routing is allowed. All d_k units of shipment k must have the same route and schedule. Shipment k must be picked up after its earliest available time at its origin ($EAT_{O(k)}^k$) and delivered before its latest delivery time at its destination ($LDT_{D(k)}^k$). The objective is to find shipment routes and vehicle and shipment schedules that minimize costs due to vehicle operations, and non-service of shipments. We consider early drop-offs to have no bonuses, and we disallow late drop-offs (that is, if a shipment is late, it will not be delivered). We are therefore interested in determining shipment routes and shipment and vehicle schedules, given the *sequence of stops* each vehicle makes. In this work, we are particularly interested in addressing the stochastic nature of input data as seen in vehicle capacities, demands of shipments, and service times.

SRTW-UU is at the core of the classic network design problem of vehicle routing with pickup and delivery of shipments under time-windows *and under uncertainty*, denoted VRPPDTW-UU [9]. The VRPPDTW-UU reduces to the SRTW-UU if we assume the routes of vehicles to be known, with the schedule still unspecified. The approach we develop for SRTW-UU problems is directly applicable to MCFTW-UU problems, and provides a natural foundation on which to develop approaches for problems involving network design and resource allocation under uncertainty.

Instances of the SRTW-UU arise in package delivery, container scheduling, airline scheduling, etc. These problems have been shown to be NP-hard [9]. Across these different applications, approaches to capture uncertainty and build robust solutions have been in three categories: (i) tailored approaches, (ii) general, distribution-free approaches, and (iii) general-distribution-based approaches. Tailored approaches like [17], [11], [15] and [10] identify specific features of the problem that can make the solution flexible, and maximize such attributes. General, distribution free approaches do not capture distribution information, but use information about uncertainty sets, as described in [18], [2], [3], [5] and [4]. General, distribution-based approaches such as [6], [7], [8], [16] and [14] model the underlying distributions analytically or through scenarios, to generate robust solutions for those distributions. We are interested, in this

work, to be able to make use of data distributions, if known, or partial information, if available; for example, in the form of quantiles.

1.2 Motivation for a New Approach

Existing optimization models for the SRTW typically model the problem in two ways: (i) using a static network, where the temporal decisions are captured as continuous variables, and (ii) using a dynamic, time expanded network, where time is captured through the network structure. When modeling uncertainty, formulation (i) has a structure that is not amenable to column generation [12]. Formulation (ii) requires making copies of arcs to capture uncertainty in time, and can present a formidable challenge in terms of the number of possible paths on the network. The challenge, therefore, is to design a model structure that can incorporate data related to parameter uncertainty, but remain tractable.

1.3 Contributions

In addressing the problem of SRTW-UU, our contributions are as follows. We develop a decomposition scheme that provides a new modeling and algorithmic approach and provides robust solutions that are less vulnerable to uncertainty. Our approach: (i) provides a new way of modeling the problem by separating the routing and scheduling elements of the problem, and maintaining accuracy by iterating among the modules; (ii) is flexible with respect to data requirements, and (iii) is applicable to large-scale problems while remaining tractable.

1.4 Outline of the paper

In §2 we present our new decomposition modeling approach. We present the different elements of the model and discuss how uncertainty is captured using this model. In §3 we present an algorithm to solve the decomposition model. We discuss the advantages and limitations of the approach. In §4 we apply our approach to the scaled down problem of a U.S. carrier, and discuss results. We summarize and conclude in §5.

2 Decomposition Modeling Approach

2.1 Decomposition Overview

Our decomposition approach for SRTW-UU involves the repeated solution of a master problem and sub-problems. At each iteration of the procedure, the master problem is solved to generate a proposed solution to the SRTW-UU problem. Each solution to the master problem ensures satisfaction of all constraints in the problem except scheduling constraints; and minimum cost with respect to the satisfied constraints. In the master problem, scheduling constraints are relaxed and schedule infeasibilities can exist in the master problem solution. To test whether or not such infeasibilities exist, we solve commodity-specific sub-problems in which infeasibilities are detected using

efficient network node-labeling algorithms. If no infeasibilities are found, a feasible schedule exists for the MCFTW-UU solution and the MCFTW-UU problem is solved. If, however, scheduling conflicts are identified, these scheduling conflicts are translated into inequalities that are added to the master problem to eliminate the current infeasible solution. After a finite (but possibly large) number of iterations, our approach is guaranteed to find a feasible, and hence optimal, solution to the SRTW-UU problem, as discussed in §3.6.

2.2 Flow Master Problem (SRTW-UU-MP)

When scheduling constraints are relaxed, the SRTW-UU reduces to flowing shipments on the vehicles. In the absence of uncertainty, the flow master problem involves solving the standard path-based multi-commodity flow formulation detailed in Ahuja et al. [1], that is, choosing paths for each k on its network G_k . Here, we present directly the notation and formulation to model uncertainty in demands and capacities using the *Extended Chance-Constrained model* (ECCP) presented in Marla [13], under a robustness budget Δ . In the ECCP approach, we know partial information about the distributions of uncertain parameters, in the form of quantiles; or full information in the form of distribution parameters. Among these different levels of protection, the ECCP approach allows the model to maximize the level of protection under a robustness budget. This allows costs to be contained when achieving robustness, and avoids the necessity for the user to specify *a priori* protection levels, which can be difficult when multiple uncertain parameters in several constraints are involved [12]. For details of the approach, we refer the reader to [13]. In the SRTW-UU, we capture uncertainty explicitly in capacities, and through the increased slack in the capacity constraint, also protect against demand uncertainty.

For each shipment k we build a network $G_k = (N_k, A_k)$ that is a copy of network $G = (N, A)$ described here. In $G = (N, A)$, each node $j \in N$ has three attributes: a location $l(j)$, vehicle $v(j) : v(j) \in V$, and information if it represents arrival or departure of v at $l(j)$. Connecting these nodes are arcs $a \in A$ of three types: travel arcs, connection arcs and transfer arcs. Travel arcs (i, j) on the network represent movement of vehicle $v(i)(= v(j))$ departing from $l(i)$ and arriving at $l(j)$. Flows on these arcs represent the movement of shipments on $v(i)$ from $l(i)$ to $l(j)$. Connection arcs connect the arrival node of $v(i)$ at location $l(i)$ and the departure node of $v(i) = v(j)$ at $l(j) = l(i)$. Flows on these arcs represent shipments remaining on $v(i)$ while it is positioned at $l(i)$. Travel and connection arcs belong to A_v , the set of arcs of vehicle v . Flows on *transfer arcs* (i, j) , which connect the arrival node of $v(i)$ at $l(i)$ to the departure node of $v(j)(\neq v(i))$ at $l(j) = l(i)$, represent the transfer of shipments between vehicles, and has an associated transfer time. In addition, we have an *artificial arc* for each shipment $k \in K$, connecting the departure node of the shipment's origin location, denoted $O(k)$, to the arrival node of its destination location, denoted $D(k)$. Artificial arcs have infinite capacity and zero travel time, but have large cost equal to that associated with non-service of shipments. $G = (N, A)$ is used to aggregate information (detailed in §3) from the networks $G_k = (N_k, A_k) \forall k \in K$.

c_{ij} is the cost on arc (i, j) . c_p is the cost due to 1 unit of flow on path p , equivalent to the sum of costs of the arcs on the path. $c_p = \sum_{(i,j) \in p} c_{ij}$. P^k is the set of origin to destination paths for commodity k in G_k . δ_{ij}^p is an arc-path indicator variable

that is equal to 1 if arc (i, j) is on path p , 0 otherwise. $f_p'^*$ is the optimal solution to the deterministic SRTW-UU-MP that minimizes costs when data assume nominal values. u_{ij} is the capacity of arc (i, j) , indicating vehicle capacities or transshipment capacities. Q_{ij} is the set of quantiles $q = 1, \dots, |Q_{ij}|$ of uncertain capacity parameters u_{ij} , for each constraint corresponding to arc (i, j) . u_{ij}^q is the capacity associated with quantile $q \in Q_{ij}$ and p_{ij}^q is the protection level probability associated with quantile $q \in Q_{ij}$ for the capacity constraint corresponding to arc (i, j) , $0 \leq p_{ij}^q \leq 1$; such that $P(u_{ij} \leq u_{ij}^q) = p_{ij}^q$. The goal of our model is to choose the solution with the highest level of protection, within a budget. Variables f_p equal 1 if all d_k units of commodity k flow on any path $p \in P_k$; and equal 0 otherwise. y_{ij}^q is the binary variable that is equal to 1 if the protection level expressed as a probability p_{ij}^q , represented by the q th quantile, is attained in the capacity constraint for arc (i, j) ; and 0 otherwise.

SRTW-UU-MP:

$$\max \sum_{(i,j) \in A} w_{ij} \gamma_{ij} \quad (1)$$

$$\text{s.t.} \sum_{k \in K} \sum_{p \in P^k} d_k c_p f_p \leq \sum_{k \in K} \sum_{p \in P^k} d_k c_p f_p'^* + \Delta \quad (2)$$

$$\sum_{p \in P^k} f_p = 1 \quad \forall k \in K \quad (3)$$

$$\sum_{k \in K} \sum_{p \in P^k} d_k f_p \delta_{ij}^p \leq \sum_{q=1}^{|Q_{ij}|} u_{ij}^q (y_{ij}^q - y_{ij}^{q-1}) \quad \forall (i, j) \in A \quad (4)$$

$$y_{ij}^q \geq y_{ij}^{q-1} \quad \forall q \in Q_{ij}, \forall (i, j) \in A \quad (5)$$

$$y_{ij}^0 = 0 \quad \forall (i, j) \in A \quad (6)$$

$$y_{ij}^{|Q_{ij}|} = 1 \quad \forall (i, j) \in A \quad (7)$$

$$\gamma_{ij} \leq \sum_{q=1}^{|Q_{ij}|} p_{ij}^q (y_{ij}^q - y_{ij}^{q-1}) \quad \forall (i, j) \in A \quad (8)$$

$$f_p \in \{0, 1\} \quad \forall p \in P^k, \forall k \in K \quad (9)$$

$$y_{ij}^q \in \{0, 1\} \quad \forall q \in Q_{ij}, \forall (i, j) \in A \quad (10)$$

$$0 \leq \gamma_{ij} \leq 1 \quad \forall (i, j) \in A \quad (11)$$

$$\text{Cuts from the Scheduling Sub-Problem.} \quad (12)$$

(1) is the ECCP objective function that maximizes a weighted sum of protection levels (with weights w_{ij} and achieved protection γ_{ij}) over constraints with uncertain parameters. Constraints (2) limit the expected cost of the robust solution to no more than a user-specified budget of Δ more than the expected optimal cost when using nominal parameter values. Constraints (3) assign one path to each shipment. Constraints (4) find the highest protection level attainable for the uncertain parameters. Inequalities (8) set γ_{ij} to be no greater than the highest protection level provided to the capacity constraint corresponding to (i, j) . (5) ensure that the protection level variables follow a step function, that is, if a higher level of protection is achieved, all

lower levels of protection are also achieved. (6) and (7) set the boundary values of the step functions. Constraints (9), (10) and (11) describe the variable ranges.

2.3 Scheduling Sub-Problem (SRTW-UU-SP)

Given a shipment flow solution F from the SRTW-UU-MP, the objective of the Scheduling Sub-Problem (SRTW-UU-SP) is to determine if the shipment flows obtained from solving the Flow Master Problem have a feasible schedule obeying time-windows and allowing connections and transfers. If a shipment is assigned to any path other than the artificial arc (in which case it is not possible to deliver it) in the solution to (??) - (??), it is possible that infeasibility might exist in its schedule. We determine if there exists a feasible schedule for the Flow Master Problem by solving a series of shortest path and network-labeling algorithms, detailed in §3.1 and §3.3. If we find a feasible schedule, we are done, otherwise, the same algorithm identifies the sources of infeasibility.

To capture uncertainty in service times, we use a priori defined quantile values of service time (travel time, transfer time, connection time) on each arc. These quantile values, representing the protection levels we want to provide, provide additional slack in the service times, thereby acting as ‘buffers’. The quantile value for the *arc* service time can be determined based on the desired protection level for the *path*, using our knowledge of uncertainty based on historical data. The actual quantile level can be changed during the iterations of the decomposition approach. During subsequent applications of the sub-problem to the master problem solution, higher quantile values may be used if a higher level of protection for the service times is desired.

We provide here a bound to help choose the a priori quantiles for paths obtained from the Master Problem. Consider a shipment path $j_0 - j_1 - j_2, \dots, -j_n$ containing n arcs. Let t_{ij} for all $(i, j) \in A$ be the service time on arc (i, j) . If t_{ij} for all $(i, j) \in A$ are independent random variables, and T_{ij} for all $(i, j) \in A$ represent, say, the α th quantile values of the arc service times, such that $\Pr(t_{j_l j_{l+1}} < T_{j_l j_{l+1}}) = \alpha$, for all $l = 0, \dots, n-1$, and:

$$\begin{aligned} \Pr(t_{j_0 j_1} < T_{j_0 j_1}, \dots, t_{j_l j_{l+1}} < T_{j_l j_{l+1}}, \dots, t_{j_{n-1} j_n} < T_{j_{n-1} j_n}) \\ = \prod_{p=0}^{n-1} \Pr(t_{j_p j_{p+1}} < T_{j_p j_{p+1}}) = \alpha^n, \text{ and} \end{aligned} \quad (13)$$

$$\Pr\left(\sum_{l=0}^{n-1} t_{j_l j_{l+1}} > \sum_{l=0}^{n-1} T_{j_l j_{l+1}}\right) \leq 1 - \alpha^n. \quad (14)$$

This provides a weak bound on the level of protection of the service time along the path $j_1, j_2, \dots, j_n$. Far stronger bounds can be obtained for some simple assumptions, such as uniform or gaussian distributions of service times.

2.4 Iterative Feedback Mechanism

We iterate between solving the SRTW-UU-MP and the SRTW-UU-SP, identifying infeasibilities in the SRTW-UU-SP solutions and adding them as new constraints into the SRTW-UU-MP in order to eliminate current infeasibilities. With each iteration, the constraints added to the SRTW-UU-MP increase its size minimally, but decrease the size of its feasible solution space. Thus, we converge to the optimal solution with each

iteration of the algorithm. We have solved the SRTW-UU when the shipment flows obtained from the SRTW-UU-MP have associated feasible schedules, and the iterative procedure terminates.

3 Solution algorithm for the decomposition modeling approach

3.1 Step 1: Network Pre-processing

On G_k , for all $k \in K$, we define EAT_j^k as the earliest time shipment k can reach node j after starting from $O(k)$ and LDT_j^k as the latest time shipment k can leave node j to get to its destination $D(k)$ on time. $sp_{i,j}^k$ is the shortest path distance for shipment k from i to node j . EAT_j^v and LDT_j^v are the earliest arrival time and latest departure time of vehicle v at each node $j \in N$.

The steps of the network Pre-processing phase are detailed as follows:

- (i) For each shipment network G_k for all $k \in K$, based on $EAT_{O(k)}^k$ and $LDT_{D(k)}^k$, find time-windows, expressed as the earliest arrival time and latest departure time $[EAT_i^k, LDT_i^k]$, of any shipment $k \in K$ at each node $i \in N_k$. Thus,

$$EAT_i^k = EAT_{O(k)}^k + sp_{O(k),i}^k; \quad \text{and} \quad LDT_i^k = LDT_{D(k)}^k - sp_{j,D(k)}^k \quad (15)$$

- (ii) Find time-windows (EAT_i^v, LDT_i^v) for each vehicle $v \in V$ at each node $i \in N$ along its route based on the earliest start time and latest allowed return time at the depot. These constraints are driven by driver work rules. If vehicle v does not pass through a node i , i is labeled unreachable for v . For all reachable arcs $(i, j) \in A_v$, for all $v \in V$, we have

$$EAT_j^v = EAT_i^v + tt_{ij}; \quad \text{and} \quad LDT_i^v = LDT_j^v - tt_{ij} \quad (16)$$

We label the time-windows and labels of vehicles and shipments thus obtained as *pre-processing time windows or labels*. They are the broadest set of time windows possible over any travel path for the vehicles and shipments (because these time-windows are formed using the *shortest* paths).

- (iii) If $LDT_i^k - EAT_i^k < 0$, the time-window duration of a shipment k is negative at a node $i \in N_k$. Because the pre-processing time-windows are the broadest time-windows, no schedule feasible path for shipment k passes through i . For each shipment network $G_k \forall k \in K$, remove all the schedule infeasible nodes, and all arcs incident to these nodes. This results in a reduced version of G_k containing only those arcs and nodes through which shipment k may pass.
- (iv) We say that the time-windows of shipment k and vehicle v overlap at node i if the combined *vehicle-shipment pre-processing time-window* has non-negative duration. We denote the time-window at node i for the vehicle-shipment pair v and k as: $(EAT_i^{k,v}, LDT_i^{k,v})$, where $EAT_i^{k,v} = \max\{EAT_i^k, EAT_i^v\}$ and $LDT_i^{k,v} = \min\{LDT_i^k, LDT_i^v\}$. Find all possible overlaps of shipment-vehicle pairs at each node $i \in N, \forall k \in K, \forall v \in V$. If the overlap between time-windows of shipment k and vehicle v is negative at node i , that is, $LDT_i^{k,v} - EAT_i^{k,v} < 0$, shipment k cannot travel on the vehicle arcs in A_v incident to node i . We delete such arcs and nodes from G_k , further reducing its size. If the time-windows of a vehicle-shipment pair (v, k) are non-zero at both ends of an arc $(i, j) \in A_v$, shipment k

can travel on (i, j) , that is, on that segment of vehicle v 's path, within the specified time-windows.

- (v) Consider the aggregate network G with the vehicle-shipment pre-processing time-windows for all $k \in K$ superimposed on each other at each node $i \in N$. Suppose the time windows of two shipment-vehicle pairs (v, k_1) and (v, k_2) at nodes i and j on the ends of arc (i, j) , are individually positive; but do not overlap with each other, thus making it infeasible for both k_1 and k_2 to travel together on $(i, j) \in A_v$ in any schedule-feasible solution to the SRTW-UU. This infeasibility can be eliminated from the set of feasible solutions to the Flow Master Problem by adding the following constraint:

$$\sum_{p_1 \in P_{k_1} | (i,j) \in p_1} f_{p_1} + \sum_{p_2 \in P_{k_2} | (i,j) \in p_2} f_{p_2} \leq 1, \quad (17)$$

where $f_p, p \in P_k$ is defined as in (1)-(12), that is, it is a binary variable that takes on value 1 if shipment k is assigned to path p ; and 0 otherwise. We add these constraints to the Flow Master Problem in the Pre-processing step to eliminate known infeasible solutions.

3.2 Step 2: Flow Master Problem

The goal of the Flow Master Problem is to assign a route to each shipment in the network. We formulate the basic route choice without uncertainty as a multi-commodity flow problem of choosing paths on the networks G_k (resulting after the Pre-processing step), for each shipment in K . To capture uncertainty in capacities and demands, we use the Extended Chance-Constrained Programming (ECCP) [13] that results in the SRTW-UU-MP. The ECCP maintains the structure of the multi-commodity flow problem and allows the use of implicit or explicit column generation techniques for large instances [12]. Other approaches that change the structure of the multi-commodity flow formulation, such as Bertsimas and Sim's robust framework [5] [4], can also be used, and explicit column generation may have to be used for large-scale instances.

3.3 Step 3: Scheduling Sub-problem (SRTW-UU-SP)

After solving the SRTW-UU-MP, vehicle routes and assigned shipment paths $p'_1, \dots, p'_{|K|}$ are known. Though the paths introduced into the Flow Master Problem are individually schedule-feasible, it is still possible that interactions between these shipment paths $p'_1, \dots, p'_{|K|}$ produce infeasible schedules. Therefore, in the Scheduling Sub-Problem, we 'flow' these shipments on the network G to determine a combined feasible schedule. Thus all the computations in this step are on the aggregate network.

- i) Initialization: Let $k = 0$ represent the vehicle flows, and commodities $k = 1, \dots, |K|$ represent the shipments.
 - (a) Set $EAT_i^k = 0$, and $LDT_i^k = M$, a very large number, $\forall i \in N, \forall k \in K$.
Set $EAT_{O(k)}^k$ to EAT_k at its origin, $LDT_{O(k)}^k = M$, $LDT_{D(k)}^k$ to LDT_k at its destination, and $EAT_{D(k)}^k = 0$.
 - (b) Set $EAT_i = 0$, and $LDT_i = M$.
 - (c) Set processing list to empty.

- (d) For $k = 0, 1, \dots, |K|$, determine the time-windows for shipment k in sequence along its currently assigned path p'_k , (forwards from $O(k)$ for EAT values and backwards from $D(k)$ for LDT values) in the aggregate network, as indicated in (18) and (19).

$$EAT_j^k = \max\{EAT_j^k, EAT_i^k + tt_{ij}\} \quad \forall (i, j) \in p'_k, \forall k = 0, 1, \dots, |K| \quad (18)$$

$$LDT_i^k = \min\{LDT_i^k, LDT_j^k - tt_{ij}\} \quad \forall (i, j) \in p'_k, \forall k = 0, 1, \dots, |K| \quad (19)$$

These time-windows will at least be as tight as the Pre-processing step time-windows, because each shipment $k \in K$ is restricted to path p'_k . The time-windows for the vehicles ($k = 0$) remain the same as those in the Pre-processing step, because the vehicle routes are given inputs that do not change in solving the SRTW-UU.

- ii) For node $i \in N$ and $k = 0, 1, \dots, |K|$,
 (a) if $EAT_i^k > EAT_i$ and $EAT_i^k \neq 0$ set $EAT_i = EAT_i^k$,
 (b) if $LDT_i^k < LDT_i$ and $LDT_i^k \neq M$ set $LDT_i = LDT_i^k$,
 (c) and add k to the processing list, if it not already present.
 We refer to (EAT_i, LDT_i) as the *movement time windows* at node i .
 iii) If the processing list is not empty, remove the first element from the list, and go to step iv, else go to step v. If the processing list is empty, it indicates that the widest movements time-windows allowing all the flows that are output from the Flow Master Problem, are found.
 iv) Update EAT_i, LDT_i for all $(i, j) \in p'_k$ (that is, for each arc in each shipment path chosen by the Master Problem), processing (i, j) in sequence along p'_k as:

$$EAT_j = \max\{EAT_j, EAT_i + tt_{ij}\}, \quad LDT_j = \min\{LDT_j, LDT_i - tt_{ij}\} \quad (20)$$

Return to Step (iii).

- v) Define $l_{ij} = LDT_j - EAT_i, \forall (i, j) \in A$. Find the shortest, non-cyclic path p^* , beginning and ending at any node, in the aggregate network with arc lengths l_{ij} . Suppose the length of p^* is l^* . If path p^* has negative length, that is, $l^* < 0$, let K^* be the set of commodities assigned to at least one arc in p^* ; otherwise stop - a feasible schedule has been identified.
 vi) Add the following constraint to the Flow Master Problem:

$$\sum_{k \in K^*} f_{p'_k} \leq |K^*| - 1; \quad (21)$$

with $|K^*|$ representing the size of the set K^* . Constraint (21) states that the set of paths causing schedule infeasibility of the current solution should not be repeated in further iterations.

3.4 Stopping Criterion

If no infeasibilities are identified in the scheduling algorithm, that is, $l^* \geq 0$ the SRTW-UU is solved; otherwise, the algorithm returns to Step 2 with added constraints of the type (21) in the Flow Master Problem. We iterate between solving the Flow Master Problem and the Scheduling Sub-Problem until the algorithm terminates with a feasible routing and schedule.

3.5 Output

The solution obtained from the above algorithm is a set of paths to which shipments are assigned and a set of *time-windows* indicating the earliest and latest time each vehicle and shipment movement can occur. The solution might route a shipment along one or more vehicles; or on its artificial arc, in which case it is not served and incurs a penalty. The schedule time-windows for the routing provide bounds within which the current set of vehicle and shipment flows may be scheduled. In the case when schedule delays and disruptions occur, the windows allow the solution to remain feasible at times, even when re-scheduling is necessary. This is indicative of the *inherent flexibility* provided by the solution to our decomposition approach.

3.6 Convergence of the Algorithm

We add a note on the convergence of our algorithm. In each iteration of our decomposition approach, we solve a relaxed version of the SRTW-UU formulation in the SRTW-UU-MP by relaxing scheduling constraints. Thus the cost incurred by the solution to the SRTW-UU-MP is a lower bound on the objective function cost of the SRTW-UU. As we add cuts to the Flow Master Problem, the objective function cost increases or stays the same. Each cut corresponds to the elimination of at least one infeasible solution to the SRTW-UU. Hence, each cut is unique, and after a finite (but possibly large) number of iterations, all infeasible solutions are eliminated and the Flow Master Problem solution will be feasible to the SRTW-UU. Because the Flow Master Problem is a relaxation of SRTW-UU and the added cuts eliminate only infeasible SRTW-UU solutions, finding a feasible schedule to the Flow Master Problem corresponds to solving, that is, finding an optimal solution to, the SRTW-UU.

The proofs for the correctness of the decomposition algorithm are not presented in this paper for the sake of brevity, but are available in Marla [12].

3.6.1 Advantages of the Decomposition Approach

We summarize the advantages of our decomposition approach as:

- The decomposition methodology involves solving a multi-commodity Flow Master Problem and a series of network-based, easy-to-solve sub-problems. Because we are breaking a large optimization problem into two smaller parts, one involving finding an optimal solution and the other simply finding a feasible solution, tractability is enhanced.
- The Flow Master Problem is a multi-commodity flow problem on a reduced-size, static network; augmented using the ECCP model, and with constraints from the Pre-processing and Scheduling Sub-Problems. Thus its size is more tractable compared to a traditional time-expanded network. In addition, as the underlying structure of multi-commodity flow is preserved in the Flow Master Problem, standard solution techniques can be applied.
- The Scheduling Sub-Problems to ascertain whether or not a feasible schedule exists for the Master problem solution are very efficient. Polynomial-time network labeling algorithms are used, which not only identify if a feasible schedule exists, but also identify of one or more constraints that can be added to the master problem to guide it towards schedule-feasible solutions.

- Capturing uncertainty in the decomposition approach does not incur much additional complexity relative to solving the SRTW for the nominal case.
- Specific types of additional constraints, such as those restricting time-windows of a vehicle at a particular location, can be added without increasing algorithmic complexity.

3.6.2 Limitations of the Decomposition Approach

We summarize the limitations of our decomposition approach as follows:

- Correlations between uncertain parameters are not captured. For example, it is possible that travel times on adjacent arcs on the network are correlated. As yet, we cannot capture such correlations.
- The choice of ‘protection levels’ for travel times on the network has to be made a priori, and involves repeated solution of instances of our model. It is possible to apply a sequential process to try increased protection levels in later iterations of the Scheduling Sub-Problem, however, it would involve a trial-and-error process on the part of the user. As in the case of applying the ECCP to the Flow Master Problem, it would be useful to develop a mechanism to automate the choice of protection levels.

4 Proof of Concept

We apply our approach to a scaled-down version of a large U.S. carrier’s shipment routing problem with time windows and uncertainty in connection times. The problem instance contains 5 locations, 3 vehicles and 6 shipments. Each vehicle has a capacity of 2 units. Each arc has a travel time of 1 unit and connections and transfers take 0.2 units in the nominal case. Costs are associated with non-service of shipments. In this example, we focus on uncertainty in service times, and in particular, we assume uncertainty to exist only in the connection times. Note that this assumption incurs very little loss of generality because uncertainty in travel times can similarly be captured by ‘shifting’ the uncertainty onto the connection times. We assume that the connection times all vary uniformly between values of 0 and 0.4 time units. The value used for the nominal case is the commonly used expected value, that is, 0.2.

We solve this problem using three approaches: (i) traditional deterministic ‘static’ approach, (ii) decomposition approach without uncertainty and (iii) decomposition approach with uncertainty. In the robust decomposition approach, we choose a protection level of 75%, yielding a value of 0.3 for the service time. The solutions obtained from each approach are evaluated on a set of randomly generated scenarios, with the realization of each of 100 scenarios constructed by sampling from a uniform distribution between 0 and 0.4. The results from the approaches are summarized in Table 1.

For the nominal case, we observe that the optimal schedule from the conventional approach is contained within the time-windows obtained by our decomposition approach, that is, they formed a proper subset of all solutions. Because the solution to the decomposition approach is in terms of time-windows, it automatically indicates the extent to which the schedule can be changed while keeping the current routing solution feasible, and thus it is feasible for 73% of the sampled scenarios. In contrast, the conventional approach, with its exact points-in-time schedule, was feasible for only 18% of the sampled scenarios.

	Deterministic solutions		Robust solution
	Static	Decomposition	Decomposition
Solution Nature	Shipment paths, exact times	Shipment paths, time-windows	Shipment paths, time-windows
Cost	0	0	0
% scenarios feasible	18	73	95

Table 1 Computational Results of the Decomposition Approach on a Small Instance

The robust decomposition approach, with 75% protection level results in a solution that is not vulnerable to uncertainty in 95% of the scenarios and requires replanning about 5% of the time. Even with a fairly low quantile level of protection of 75%, the solution's robustness increased compared to the deterministic decomposition approach and very significantly compared to the static deterministic approach. This simple example further points to the inherent robustness provided by the solution structure of the decomposition approach, which allows for dynamic flexibility.

4.1 Applicability to Large-scale instances

To discuss the applicability of this algorithm to large-scale problems, we briefly discuss results from the application of this algorithm to instances of dynamic airline scheduling using data from a major US airline. However, for reasons of brevity, we do not present the full problem description and formulation here. For a full description of the problem formulation, and the model using our decomposition approach, we refer the interested reader to Marla [12].

The airline operates a hub-and-spoke network with about 1000 flights that serve approximately 100 cities on a daily basis. Partial passenger data for this network is available and is used as input. We solve the problem instance using a traditional time-space network modeling approach and using our decomposition approach. The routing and fleetling results obtained from the decomposition approach and from the conventional time-space network approach are the same, and both approaches indicated that the original flight schedule need not be re-timed. We present the size and solution times using these two approaches in Table 2. In this table, constraint matrix size is the size of the full formulation in the conventional approach, and the size of the Flow Master Problem in the decomposition approach.

	Time-Space Network	Decomposition
MIP solution time	153.49 sec	0.09 sec
Program Elapsed Time	1059.75 sec	2.56 sec
Constraint Matrix Size	185000x394000	12000x15900

Table 2 Comparison between our Decomposition Approach and a Conventional Approach

Table 3 shows the short solution times for both the Flow Master Problem and the Scheduling Sub-Problem, indicating that several iterations of the decomposition

	Solution Time
Flow Master Problem	0.1 sec
Scheduling Sub-Problem	0.01 sec

Table 3 Solution times for modules of the Decomposition approach

approach can be run in the same time that the conventional approach is solved. Addition of constraints to the Flow Master Problem as the iterations progress can increase the problem size, but it can still be manageable and tractable. In the case of large instances, our decomposition approach can potentially undergo a large number of iterations, each time creating a solution that is ‘more’ schedule-feasible, whereas conventional approaches do not give indications of progress of the solution algorithm, and therefore in very large instances, the user is unable to get any intermediate information that may be useful.

This computational experiment provides proof-of-concept that our decomposition approach might be effective in addressing large-scale problems that otherwise face issues of intractability.

5 Summary

We described a new decomposition modeling approach for shipment routing with time windows under uncertainty. The problem is decomposed into routing and scheduling modules and solved by iterating repeatedly between them. We incorporate knowledge of uncertainty in vehicle capacities and shipment demands in the routing module and service time uncertainty in the scheduling module. Preliminary computational results indicate that the output of our decomposition algorithm is ‘inherently robust’, providing windows of feasible schedules rather than schedules corresponding to points in time.

References

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms and Applications. Prentice Hall, New York (1993)
2. Ben-Tal, A., Nemirovski, A.: Robust solutions to uncertain programs. *Operations Research Letters* **25**, 1–13 (1999)
3. Ben-Tal, A., Nemirovski, A.: Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming* **88**, 411–424 (2000)
4. Bertsimas, D., Sim, M.: Robust discrete optimization and network flows. *Mathematical Programming Series B* **98**, 49–71 (2003)
5. Bertsimas, D., Sim, M.: The price of robustness. *Operations Research* **52**(1), 35–53 (2004)
6. Birge, J., Louveaux, F.: Introduction to Stochastic Programming. Springer (1997)
7. Charnes, A., Cooper, W.W.: Chance constrained programming. *Management Science* **6**(1), 73–79 (1959)
8. Charnes, A., Cooper, W.W.: Deterministic equivalents for optimizing and satisficing under chance constraints. *Operations Research* **11**(1), 18–39 (1963)
9. Cordeau, J., Laporte, G., Potvin, J., Savelsbergh, M.: Transportation on demand. *Handbook of Operations Research* **Section 2** (2006)
10. Kang, L., Clarke, J.: Degradable airline scheduling (2002). Working paper, Operations Research Center, Massachusetts Institute of Technology

11. Lan, S., Clarke, J., Barnhart, C.: Planning for robust airline operations: Optimizing aircraft routings and flight departure times to minimize passenger disruptions. *Transportation Science* **40**(1), 15–28 (2006)
12. Marla, L.: Robust optimization for network-based resource allocation problems. Masters thesis, Massachusetts Institute of Technology (2007)
13. Marla, L.: Airline schedule planning and operations: Optimization-based approaches for delay mitigation. Ph.D. thesis, Massachusetts Institute of Technology (2010)
14. Mulvey, J.M., Vanderbei, R.J., Zeinos, S.A.: Robust optimization of large-scale systems. *Operations Research* **43**, 34–56 (1981)
15. Paraskevopoulos, D., Karakitsos, E., Rustem, B.: Robust capacity planning under uncertainty. *Management Science* **37**(7), 787–800 (1991)
16. Rockafellar, R., Uryasev, S.: Optimization of conditional value-at-risk. *The Journal of Risk* **2**(3), 21–41 (2000)
17. Shebalov, S., Klabjan, D.: Robust airline crew pairing: Move-up crews (2004). Working paper, the University of Illinois at Urbana-Champaign, Available from <http://netfiles.uiuc.edu/klabjan/www>
18. Soyster, A.L.: Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research* **21** (1973)

A GRASP Heuristic to the Extended Car Sequencing Problem

Lucas M. Rizzo • Sebastián Urrutia

Abstract This paper describes a GRASP heuristic to the recently introduced Extended Car Sequencing Problem. A constructive heuristic is developed based on an heuristic for the classical Car Sequencing Problem. The local search procedure uses a very simple neighborhood easy to be evaluated. Computational results on instances from the CSPLib's library verify the efficiency of the method in comparison with the literature.

1 Introduction

The car sequencing problem (CSP) involves the schedule of different kinds of cars along an assembly line to the installation of some options as air conditioner or sunroof. Each option is installed by a station, which is capable to simultaneously deal with a limited amount of cars. The classical CSP problem [3] considers a set of cars to be produced, each of them with a set of options to be installed, and consists in finding a sequence of the cars that does not violate the limitations of any station in any part of the sequence. The problem is NP-Complete according to [2].

This paper deals with the extended version of CSP (xCSP) introduced in [1]. The extension considers not only upper limitations on the number of cars that a station can simultaneously deal with, but also lower limitations. This new limitations aims a more homogeneous distribution of work in the option installation stations.

The paper of Bautista et al. [1] is, as far as we know, the only one dealing with the xCSP. In this paper we propose a new simple heuristic to the problem based on the GRASP meta heuristic. We focus on strategies and data structures to accelerate a simple local search procedure instead of developing a complex heuristic. With that, our main goal is to reduce the computational cost of finding high quality solutions. The rest of the paper is organized as follows: Section 2 defines the problem and introduce nomenclature, operations and functions used. Section 3 describes the proposed GRASP heuristic. In section 4 the constructive heuristic used during the optimization heuristic is detailed. Section 5 describes the local search

Lucas M. Rizzo

Departamento de Ciência da Computação, Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627 - Pampulha - Belo Horizonte – MG, Brasil
E-mail: rizzo@dcc.ufmg.br

Sebastian Urrutia

Departamento de Ciência da Computação, Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627 - Pampulha - Belo Horizonte – MG, Brasil
E-mail: surrutia@dcc.ufmg.br

procedure. Finally, in section 6 the obtained results are discussed and some considerations are made.

2 The extended Car Sequencing Problem

The xCSP is defined by the tuple (C, V, p, q, r, s, t) where $C = \{c_1, c_2, \dots, c_n\}$ is the set of cars to be produced and $V = \{v_1, v_2, \dots, v_n\}$ is the set of different options to be installed. For each car c_i a subset of options needs to be installed. Vectors p and q define the maximum capacity associated to each option v_i , that is, in every sequence of $q(v_i)$ cars at most $p(v_i)$ of them may require option v_i . Analogously, vectors r and s define the minimum capacity associated with each option, that is, in every sequence of $s(v_i)$ cars at least $r(v_i)$ of them must require option v_i . Function $t: C \times V \rightarrow \{0,1\}$ is the function of requirements, for each car c_i and option v_j , $t(c_i, v_j)$ is 1 if the option v_j must be installed in car c_i and is 0 otherwise.

Below we list some notations used along the paper:

- a sequence $\pi = \langle c_{\pi_1}, c_{\pi_2}, \dots, c_{\pi_k} \rangle$ is a succession of cars.
- $|\pi|$ is the number of cars in sequence π .
- the concatenation of two sequences π_1 and π_2 called $\pi_1 \cdot \pi_2$ is the sequence of cars π_1 followed by the sequence of cars π_2 .
- the sequence π_k is a factor of another sequence π called $\pi_k \subseteq \pi$ if there are two (possible empty) sequences π_1 and π_2 tal such that $\pi = \pi_1 \cdot \pi_k \cdot \pi_2$
- the number of cars requiring option v_i in a sequence π is denoted by $t(\pi, v_i)$ and is defined by $t(\pi, v_i) = \sum_{c_k \in \pi} t(c_k, v_i)$.
- the initial position of a subsequence in the original sequence is denoted by π_{k_0} . If for example $\pi_{k_0} = 3$ we know that the subsequence π_k starts in the third position of π .
- the dynamic utilization hate of an option [2] given a sequence π is defined by $\text{dynUtilRate}(v_i, \pi) = \frac{[t(c, v_i) - t(\pi, v_i)] \cdot q(v_i)}{p(v_i) \cdot [|\pi| - |\pi_k|]}$.

Then we can define the upper over-assignment of a sequence by:

$$\text{max\_cost}(\pi) = \sum_{v_i \in V} \sum_{f(\pi, i, k)} \text{max\_violations}(\pi_k, v_i)$$

where

- $f(\pi, i, k) \pi_k \subseteq \pi$ such that $\begin{cases} |\pi_k| = q(v_i) & \text{if } \pi_{k_0} > 0 \\ p(v_i) + 1 \leq |\pi_k| \leq q(v_i) & \text{if } \pi_{k_0} = 0 \end{cases}$
- $\text{max\_violations}(\pi_k, v_i) = \begin{cases} 0 & \text{if } t(\pi_k, v_i) \leq p(v_i) \\ t(\pi_k, v_i) - p(v_i) & \text{if } t(\pi_k, v_i) > p(v_i) \end{cases}$

in other words, max_violation counts the number of cars with determined option that exceeds the upper bound $p(v_i)$ to which subsequence of size $q(v_i)$. The function max_cost sums the violations (upper violations) over every option and every subsequence of size $q(v_i)$ of the original sequence.

Analogously we define the lower under-assignment by:

$$\text{min\_cost}(\pi) = \sum_{o_i \in O} \sum_{\substack{\pi_k \subseteq \pi \text{ such that} \\ |\pi_k| = s(v_i)}} \text{min\_violations}(\pi_k, v_i)$$

where

$$\bullet \min\_violations(\pi_k, v_i) = \begin{cases} r(v_i) - t(\pi_k, v_i) & \text{if } t(\pi_k, v_i) < r(v_i) \\ 0 & \text{if } t(\pi_k, v_i) \geq r(v_i) \end{cases}$$

in other words, *min_violations* counts the numbers of cars with certain option that are missing to achieve the lower bound to each subsequence of size $r(v_i)$. The function *min_cost* sums the violations (lower violations) to each option and every subsequence of size $r(v_i)$ of original sequence.

To illustrate the problem suppose the following situation: consider a set of cars that requires the installation of only one option. We call this set c_A and the option v_a . Suppose that $p(v_a) = 2$ and $q(v_a) = 4$, in other words, every subsequence of four cars at most two of them must have the option installed. Suppose also, $r(v_a) = 1$ e $s(v_a) = 4$, then every subsequence of four cars at least one must have the option installed. Figure 1 illustrates some possible situations:

X X _ _ _ X _ X

(a) Sequence without violations

X X X _ X _ _ X



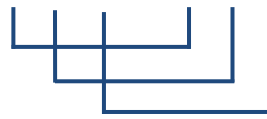
(b) Sequence with two upper violations.

X _ _ _ X X X



(c) Sequence with one upper violation and one lower violation.

X _ _ _ _ _ X



(d) Sequence with three lower violations.

Figure 1: Example of violations for a set of cars in a sequence of size eight. Cars of the set c_A are represented with a X and the rest with a empty space.

The xCSP consists in finding a sequence of the cars to be produced minimizing a weighted sum of functions *max_cost* and *min_cost*. Observe than in comparison with the classical CSP formulation the xCSP introduces the following changes: (i) includes the under assignment concept, (ii) considers an optimization problem instead of a feasibility one, and (iii) considers subsequences with size smaller than $q(v_i)$ at the beginning of the sequence.

3 Heuristic Approach

To heuristically solve the problem we implemented an algorithm based on the GRASP metaheuristic [4,6], which works as follows:

1. Final Solution $\leftarrow \emptyset$
2. **For** k=1 **to** MAXIT **do**
 - a) Solution $\leftarrow$ Randomized Greedy Algorithm
 - b) Solution $\leftarrow$ Local Search (Solution)
 - c) **If** Solution better than Final Solution
Final Solution $\leftarrow$ Solution

End-If

End-For

3. **Returns** Final Solution.

In every iteration an initial solution is constructed by a randomized greedy algorithm to be refined by a local search procedure. This process is repeated MAXIT times, updating the final solution every time a better solution is found. This final solution is returned when the algorithm terminates.

The next two sections details the two main procedures of the algorithm: The randomized greedy heuristic and the local search procedure.

4 **Randomized Greedy Algorithm**

The proposed randomized greedy algorithm is based on the heuristic algorithm proposed in [3]. It constructs a sequence of cars by adding, at each iteration, a given unscheduled car to the last position of the partial sequence. The greedy function that decides which car to add at each step considers the number of upper violations the inclusion of the car generates in the sequence being constructed. It finds the car c_j that minimizes:

$$newViolations(\pi, c_j) = \sum_{v_i \in V} t(c_j, v_i) * max\_violations(last\_cars(\pi, c_j, q(v_i)), v_i)$$

where $last\_cars(\pi, c_j, q(v_i))$ is the sequence composed by the last $q(v_i)$ cars from π, c_j if $|\pi, c_j| \geq q(v_i)$ or sequence π, c_j otherwise. Ties in this criterion are not only possible but frequent, for this case we present tie-breaking criteria:

1. Choose the car that minimizes $newViolations\_min$ (1).
2. Create a restricted candidate list by choosing two of the tied cars and choose the car that has the larger dynamic highest utilization rate which is defined by:
 $DHU(c_j) = \sum_{v_i \in V} t(c_j, v_i) * weight(v_i)$
 where $weight(v_i) = 2^k$ if v_i is the option with the lowest k dynUtilRate for the current sequence.

$$(1) newViolations\_min(\pi, c_j) = \sum_{v_i \in V} t(c_j, v_i) * min\_violations(last\_cars(\pi, c_j, s(v_i)), v_i)$$

5 **Local Search**

The swap neighborhood is defined by a swap operator which consists in exchanging the cars in two given positions the given sequence π , that is:

$$Swap(j, i, \pi) = \pi_1 \cdot c_i \cdot \pi_2 \cdot c_j \cdot \pi_3$$

where the form of π was initially $\pi_1 \cdot c_j \cdot \pi_2 \cdot c_i \cdot \pi_3$.

Movement may only change the value of the objective function if $t(c_j, v_i) \neq t(c_i, v_i)$ for some $i \in V$.

Other neighborhoods generated by other operators are usually employed in local search procedures for the CSP. Among them we can quote insertion and inversion:

$Insertion(c_i, \pi, p) = \pi_1. \pi_2. c_i. \pi_3$ where π was the way $\pi_1. c_i. \pi_2. \pi_3$ and p the new position of c_i in π

$Inversion(\pi_1. \pi_2. \pi_3) = \pi_1. reverses(\pi_2). \pi_3$ where
 $reverses(\pi. c_j) = c_j. reverses(\pi)$

Other neighborhoods can be found in the literature. In this paper, only the swap neighborhood is used. This neighborhood is connected (all feasible solutions are reachable from any other solution by a finite number of swap movements) and is big enough (each solution has $O(n^2)$ neighbors) to allow a implementation of an efficient local search procedure.

Let us define the function *Delta* as follows: Given a upper over-assignment and a lower under-assignment of the sequence π , instead of performing the exchange between the cars c_j and c_i , *Delta* returns the variation of the upper over-assignment and the lower under-assignment if the cars c_j and c_i exchanged positions. This variation is defined by:

$$\begin{aligned} Delta(c_j, c_i) = & M * (max\_cost(\pi_1. c_i. \pi_2. c_j. \pi_3) - max\_cost(\pi)) \\ & + (min\_cust(\pi_1. c_i. \pi_2. c_j. \pi_3) - min\_cost(\pi)). \end{aligned}$$

where M is a big constant as defined in [1].

Although defined in this way, the calculating of *Delta* would be extremely inefficient if implemented explicitly computing functions *max_cost* and *min_cost*. Evaluate the cost of a entire sequence π is a very consuming process. Its computational complexity is $O(|V| * |C| * q_{max})$ where $q_{max} = \max\{q(v_i) \forall i\}$. For each of the $|V|$ options one should go over $O(|C|)$ subsequences counting, in each of them, the number of cars that requires the installation of the given option.

Note that, since the difference between two consecutive subsequences is at most two cars, after evaluating the function for the first subsequence, each next subsequence may be evaluated in $O(1)$. Thus, the computational complexity of the function decrease to $O(|V| * |C|)$. Moreover, observing that only q_{max} sequences may be affected by the swap movement the computational complexity may be further decreased to $O(|V| * q_{max})$. So we can define a function *max_cost_temp*, with complexity $O(|V| * q_{max})$, that given an upper over-assignment of a sequence π and two cars c_j and c_i to be exchanged returns the new cost considering only the subsequence where the cost may vary.

Analogously we can define *min_cost_temp* and its complexity is $O(min\_cost\_temp) = O(|V| * s_{max})$.

Then *Delta* is defined by the new way:

$$\begin{aligned} Delta(c_j, c_i) = & M * (max\_cost\_temp(K_1, \pi, c_j, c_i) - K_1) \\ & + (min\_cost\_temp(K_2, \pi, c_j, c_i) - K_2) \end{aligned}$$

where K_1 and K_2 are the upper over-assignment and the lower under-assignment respectively of the sequence to which the operation of *Swap* is being evaluated.

Finally, we propose a best improvement local search as follows:

1. **Do:**
2. **For** $i=1 \dots |C|-1$ **do:**
3. **For** $j=i+1 \dots |C|$ **do:**
4. $Delta(c_j, c_i)$
5. **End-For**
6. **End-For**

7. Find c_j and c_i such that $\Delta(c_j, c_i)$ is the lowest $\forall i, j$.
8. $\text{Swap}(j, i, \pi)$
9. **While** ($\Delta < 0$).

6 Results

All the instances used in this paper are provided by CSPLib [7] . A total of 74 instances were used in the experiments, four of them have 100 cars while the other 70 instances have 200 cars. Each instance considers five possible options to be or not installed in each car. *Table 1* shows the characteristics of the instances.

In order to adapt the instances to the xCSP we build the restrictions of minimum as in [1]:

$$s(v_i) = q(v_i) \text{ e } r(v_i) = p(v_i) - 1$$

Two parameters must be set in the proposed algorithm, the maximum size of the RCL and the constant M used in the objective function to model the relative importance of the two types of restrictions. For comparison purposes all the tests were performed with RCL=2 and rotated for a maximum of 1000 seconds for each instance as in [1] (the algorithm terminates if a solution with cost 0 is found for both upper over-assignment and lower under-assignment).

The algorithm was implemented in C and compiled using gcc. Tests were run in a Pentium IV 3.0 GHz under the Linux operating system. Both instances and machine are identical to the ones used in [1]. *Table 2* show the obtained results.

Set	Number of Instances	Solvable Instances	% use of each option	Total Cars
Classical	9	4	Variable	100
p60	10	10	60%	200
p65	10	10	65%	200
p70	10	10	70%	200
p75	10	10	75%	200
p80	10	10	80%	200
p85	10	10	85%	200
p90	10	10	90%	200

Table 1 – Characteristics of each instance. Solvable instances refer only to upper violations.

Only classical instances that are known to be solved, in other words, those for which are known a solution with zero upper violations were used in this paper. These solutions were found in [3] and [5].

Table 2 shows the computational results. The second and third columns show the number of violations of type upper over-assignment respectively for the algorithm described in this paper and the one described in [1]. The next two columns show the number of violations of type lower under-assignment. The last two columns show the number of instances solver for each set. The average time to each GRASP iteration was 0,3 seconds for the algorithm described in this paper, while in [1] an average time of 1 second per GRASP iteration is reported.

Set	UOA	UOA [1]	LUA	LUA [1]	Solvable Instances	Solvable Instances[1]
Classical	2,5	7,75	0	0	1	0
p60	0	0	11	14,3	10	10
p65	0	0	1,3	2,8	10	10
p70	0	0	0,1	0,1	10	10
p75	0	0	0,1	0	10	10
p80	0	0	0	0,1	10	10
p85	0	0,3	2,1	3,2	10	8
p90	0,3	4,2	0,8	2	7	2

Table 2 – Average costs found for each set of instances. UOA represents the Upper over-assignment and LUA the Lower under-assignment.

In view the Table 2 we can emphasize that all the found costs are better, beyond the algorithm rotates on average three times faster.

7 Conclusions

In this paper we introduced a new GRASP heuristic algorithm to the Extended Car Sequencing Problem, which outperforms the heuristic found in the literature. The constructive heuristic is an adaptation of a heuristic to the classical Car Sequencing Problem. The local search procedure is very simple using only the swap neighborhood.

A careful implementation of the local search procedure in terms of the cost evaluation of neighbor solutions allowed us to decrease the time of execution of every GRASP iteration in comparison with [1]. In consequence our algorithm finds much better solutions in the same execution time with the same computational environment.

Future works include the usage of other neighborhoods in the local search procedure and the developing of new heuristics based on other metaheuristics.

References

- [1] Bautista, J., Pereira, J. and Adenso-Díaz, B. A GRASP approach for the extended car sequencing problem, *Journal of Scheduling*, v.11, p.3-16, 2008.
- [2] Gent, I. Two results on car-sequencing problems (Technical report APES-02-1998).
- [3] Gottlieb, J., Puchta, M. and Solnon, C. A study of greedy, local search and ant colony optimization approaches for car sequencing problems. In *Lecture notes in computer science*: Vol. 2611, 246-257, 2003.
- [4] Feo T. A. and Resende M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8, 67-71, 1989.
- [5] Regin, J. C. and Puget, J. F. A filtering algorithm for global sequencing constraints. In *Lecture notes in computer science*: Vol. 1330, 32-46, 1997.
- [6] Resende, M.G.C. and Ribeiro, C.C. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications, *Handbook of Metaheuristics* (M. Gendreau e J.-Y. Potvin, editors), 283-319, Springer, 2nd edition, 2010.
- [7] <http://www.csplib.org/>

A new hyper-heuristic implementation in HyFlex: a study on generality

Mustafa Mısıř · Katja Verbeeck · Patrick De
Causmaecker · Greet Vanden Berghe

Abstract *Reusability* is a desired feature for search and optimisation strategies. Low-level, problem-dependent search mechanisms are far from being used on different problems while keeping them unchanged. Meta-heuristics have been employed for taking the heuristic design process to a higher level and for facilitating reusability. These approaches have been usually conceived as heuristic methods that need to be implemented and adapted with regard to the characteristics of the goal problem. Thus, the resulting algorithms are still problem-dependent and hard to apply to other problems. Hyper-heuristics take the search process into the heuristic level and manage the heuristic set instead of directly solving a problem. During this management process, any problem-dependent data exchange between hyper-heuristics and problems is disallowed. Although any knowledge about the problems is absent for the hyper-heuristics, the generality of hyper-heuristics has not been examined extensively. In a recently proposed high level framework, HyFlex, it is easy to test the generality of hyper-heuristics. HyFlex provides a set of problems with a number of instances as well as a group of low-level heuristics. In this study, the design process of a hyper-heuristic upon HyFlex will be discussed. A performance analysis based on the experimental results will be carried out.

1 Introduction

New problem domains or variants of existing problems have been progressively entering the literature. They draw the attention of researchers to develop effective solution strategies. The performance of the developed algorithms tends to vary over different problems or even over problem instances belonging to a particular problem. In order to alleviate this issue, algorithm selection strategies have been studied. The primary goal

Mustafa Mısıř · Katja Verbeeck · Greet Vanden Berghe
KAHO Sint-Lieven, CODES, Gebroeders Desmetstraat 1, 9000 Gent, Belgium
K.U.Leuven Campus Kortrijk, CODES, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: {mustafa.misir,katja.verbeeck,greet.vandenberghel}@kahosl.be

Patrick De Causmaecker
K.U.Leuven Campus Kortrijk, CODES, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: patrick.decausmaecker@kuleuven-kortrijk.be

of these strategies is to determine the best algorithm for the target problem instances. Even if this is a useful approach, it misses the possible improvement opportunities due to varying algorithm performance in the course of the solution process. *Selection hyper-heuristics* follow a deeper selection approach by managing a number of given low-level search strategies during the search relying on their strength and weaknesses [7].

In the literature, the number of studies concerning hyper-heuristics is exponentially growing. In [8], these hyper-heuristics were classified based on the type of the provided feedback mechanisms and the nature of the heuristic search space. Based on their feedback mechanisms, three categories are considered, namely hyper-heuristics with *online learning*, *offline learning* and *no learning*. In online learning, the learning process occurs during the search process. Choice function [16], reinforcement learning [29,31], learning automata [28] are some examples of online learning. Contrarily, offline learning refers to learning before starting the search. In particular, case based reasoning [14] and learning classifier systems [32,26] work in an offline manner. It is also available some hyper-heuristic components without any learning device. An example is the simple random heuristic selection mechanism [16]. Hyper-heuristics are categorised with respect to the nature of the given heuristics as *selection hyper-heuristics* and *generation hyper-heuristics*. The aforementioned studies handled the selection hyper-heuristics that perform over constructive or perturbative heuristics. This type of hyper-heuristics also contain various meta-heuristic components such as tabu search [13], simulated annealing [18,11], genetic algorithms [20] and ant colony optimisation [12]. On the other hand, there exist some hyper-heuristics that aim to generate the low-level heuristics. In [10,9,19,2,5], genetic programming was utilised to generate low-level heuristics designed for the problem instances.

According to the initial definition of hyper-heuristics, they have been designed to raise the level of generality [7]. Placing a domain barrier preventing any problem-dependent data transition from or to hyper-heuristics is the foremost rule for reaching generality. This basic principle leads hyper-heuristics to focus on managing low-level search strategies instead of directly solving a problem instance. Therefore, it is required to design a hyper-heuristic that has the ability to govern different heuristic sets while profiting maximally from their capabilities. HyFlex is a software framework that empowers hyper-heuristic developers to test their approaches across a range of problems [6]. In the current version, four problem domains, i.e. max SAT, bin packing, permutation flowshop and personnel scheduling, are available. Related to each problem domain, a number of perturbative heuristics that have been implemented from four main heuristic types, namely mutational heuristics, crossover operators, ruin-recreate heuristics and hill climbers. The detailed description of these problems, the characteristics and origins of instances as well as the definition of the heuristics were given in [17,22,21,35]. In addition to these features, HyFlex provides an opportunity to change the effect of mutational heuristics and hill climbers. Moreover, it is possible to keep some of the solutions in memory for further use.

In the present study, a selection hyper-heuristic is implemented on HyFlex and an experimental performance analysis on the available problems is conducted. The proposed hyper-heuristic has been developed as a general adaptive strategy for the four problem domains and the given heuristic sets. It consists of a dynamic heuristic selection mechanism and a move acceptance strategy based on the changing characteristics of the search environment. Experimental results confirm that the developed

Heuristic Type	Max SAT	Bin Packing	Perm. Flowshop	Personnel Scheduling
<i>Mutation</i>	2 (3,4)	2 (0,5)	5 (0,1,2,3,4)	1 (11)
<i>Crossover</i>	2 (7,8)	1 (6)	5 (11,12,13,14,15)	3 (8,9,10)
<i>Ruin-recreate</i>	1 (6)	2 (1,2)	2 (5,6)	3 (5,6,7)
<i>HillClimber</i>	4 (0,1,2,5)	2 (3,4)	4 (7,8,9,10)	5 (0,1,2,3,4)
<i>Total</i>	9	7	16	12

Table 1 The distribution of heuristic types for the given problem domains (heuristic indexes are shown in parentheses)

hyper-heuristic can provide significant performance improvement compared to other hyper-heuristics tested on the problem instances.

In the remainder of this paper, the generality requirements for hyper-heuristics are succinctly discussed in Section 2. The underlying components of the proposed hyper-heuristic is presented in Section 3. Next, the experimental results are discussed in Section 4. In the last section, the paper is concluded.

2 Generality requirements

A generic selection hyper-heuristic should be capable of managing a diverse range of heuristic sets utilised for solving distinct problems. Although the capability of solving as many problems as possible is the primarily mentioned focus, the main concern should be the management of different heuristic sets. This objective implicitly embraces the aim of solving various problems anyway. The characteristics of the existing low-level heuristics may require distinct management strategies because each heuristic may have various advantages and disadvantages. These features should be interpreted relying on the dynamic performance of the heuristics and experimental limits such as the given execution time. And the heuristic set as a whole should be used in synergy. For this purpose, a heuristic selection mechanism should consist of particular analysis components to facilitate the adaptivity of the selection process.

2.1 Heuristic set feature

An analysis tool or a learning component should be designed based on a set of characteristics that determine the behavior of the heuristics. The first characteristic is related to the heuristic *set size*. A heuristic set with many heuristics has a higher probability of finding satisfactory solutions regarding a problem instance. Conversely, such a set can be hard to manage due to the availability of many options to select. The quality of heuristics is also critical about the set size. If the heuristics have similar performance, then the set size can make no difference. Since this option is uncommon, it is required to employ effective learning strategies. The second feature is the *speed* of the heuristics. This characteristic may affect the number of decision steps for the selection process. Hence, it is required to interpret this element with the *improvement capabilities* of the heuristics. The heuristic *specialisation* is another factor. A heuristic dedicated to solve a constraint or improve an objective can be considered in this category. In addition to that, heuristics generating only improving or equal quality solutions such as hill climbers or any other strict heuristic behavior come in this category. This feature list can obviously be extended. The main purpose should be the usage of the most relevant

and effective features [1, ch.6] in a collaborative way to predict the future performance of the heuristics.

2.2 Parameter and rule settings

Parameter-free strategies are interesting from a generality purpose perspective. Even though such methodologies are called parameter-free, their behaviour depends on some predetermined values or rules. For instance, simple random is a parameter-free heuristic selection mechanism that gives an equal selection chance to each heuristic. The simple random is parameter-free since its parameters are set from the beginning. Another example is the improving or equal move acceptance mechanism. It accepts only better or equal quality solutions. This method is based on a predetermined rule. That is, the move acceptance method is parameter free, yet not rule-free. All similar methods show that providing a totally independent mechanism seems impossible. Instead, the proposed algorithms should be able to control their parameters or rules according to the search space and the environmental settings with the aim of decreasing the user effect on the algorithm's performance.

3 Hyper-heuristic

A traditional selection hyper-heuristic requires a selection mechanism to determine the best heuristic to apply at each decision step. In addition, it needs a move acceptance strategy to check whether the constructed/visited solution is accepted with regard to its quality and the current state of the search. These consecutive operations are performed until the stopping condition is met. In this study, a new heuristic selection mechanism and a move acceptance strategy including additional components are proposed. In the following paragraphs, these submechanisms are explained in detail.

3.1 Heuristic selection

3.1.1 *Dynamic heuristic set*

In [27,34], a dynamic heuristic set strategy for the heuristic selection process was studied. The motivation behind this approach is determining elite heuristic subsets during specific iteration intervals. Similar approaches aiming to eliminate heuristics were studied in [13,15,23,24]. The dynamic heuristic set strategy was carried out by eliminating heuristics that are expected to perform worse compared to the rest and keep the best ones determined via some performance criteria. These criteria reflect the performance changes of the heuristics during the search. Performance changes are determined based on the number of new best solutions found, the total fitness improvement and disimprovement per unit execution time. These elements are used according to their importance for the search process. For instance, finding a new best solution is more important than improvement without a new best solution. That is, a heuristic that reaches a new best solution is considered as a higher quality heuristic. This information was gathered during a phase composed of a predetermined number of iterations. In the new selection strategy, i.e. adaptive dynamic heuristic set (ADHS), an updated version of this performance metric is proposed.

A weighted sum of different performance elements was used to determine the quality of different search strategies. In equation (1), the details of the performance measurement for each heuristic is given. In this equation, $C_{p,best}(i)$ denotes the number of new best solutions found. $f_{imp}(i)$ and $f_{wrs}(i)$ show the total improvement and worsening provided. $f_{p,imp}(i)$ and $f_{p,wrs}(i)$ refer to the same measurement be it in a particular phase only. t_{remain} refers to the remaining time to finish the whole search process. $t_{spent}(i)$ and $t_{p,spent}(i)$ are the time spent by heuristic i until now, the same measurement during a phase respectively. And each w_i denotes the weight of a performance element. The weights are set as $\{w_1 \gg w_2 \gg w_3 \gg w_4 \gg w_5\}$ to provide a strict priority between the given performance elements. Thus, for instance, if the first performance element of a heuristic has the highest value, than the rest of the performance elements have no effect on the overall p_i of the corresponding heuristic.

$$p_i = w_1 \left[\left(C_{p,best}(i) + 1 \right)^2 \left(t_{remain} / t_{p,spent}(i) \right) \right] + w_2 \left(f_{p,imp}(i) / t_{p,spent}(i) \right) - w_3 \left(f_{p,wrs}(i) / t_{p,spent}(i) \right) + w_4 \left(f_{imp}(i) / t_{spent}(i) \right) - w_5 \left(f_{wrs}(i) / t_{spent}(i) \right) \quad (1)$$

In this performance criterion, the first performance element is related to the heuristic's capability of finding a new best solution. In the aforementioned studies, this element was just a counter of new best solutions found by a heuristic. However, it may cause some difficulties especially if some heuristics can find new best solutions only after a long time. For that reason, it can be useful to apply a heuristic that may find more new best solutions during the given execution time. In addition to that, since this strategy works based on the exclusion of heuristics, it can be useful to have plus 1 for the first part. Otherwise, a relatively slow heuristic can stay in the heuristic set regardless of its speed because of finding new best solutions. The second element is used to select improving heuristics and the third element is employed to choose heuristics that deteriorate solutions less. The last two elements have a similar aim, but their values are independent from phases. They are calculated using the values collected until that moment.

The number of phases where a heuristic stays out of an elite heuristic set is denoted by tabu duration. For decreasing user dependency, the tabu duration and the number of iterations for one phase are calculated based on the number of heuristics available in the elite heuristic subset. The tabu duration is set to $d = \sqrt{2n}$ and the phase length (pl) is defined as the product of the tabu duration and a constant value ($ph_{factor} = 500$). These values are recalculated at the end of each phase. Since the calculated p_i values are a combination of different quantities and they are noisy, the performance values are converted into a quality index ($QI \in [1, n]$) value. A heuristic with the lowest p_i gets 1, the others get one unit more based on their order. The average (avg) of these QI values is calculated to determine the heuristics that will be excluded. Tabu heuristics are also attend this calculation with $QI = 1$.

$$avg = \left\lfloor \left(\sum_i^n QI_i \right) / n \right\rfloor \quad (2)$$

Tabu duration adaptation: The tabu duration of a specific heuristic is increased if it is successively excluded. In other words, the tabu duration is specifically determined for

each heuristic using d as the based value. This is required since the proposed exclusion procedure gets a tabu heuristic back whenever its tabu status expires.

Phase length adaptation: The number of iterations per phase is determined as $pl = d * ph_{factor}$. However, this value is updated based on the speed of the non-tabu heuristics as whole for fairness. For that purpose, at the end of each phase, the spent time per move concerning each non-tabu heuristic is calculated and the total of all (t_{subset}) is used to determine the next phase length. In the following equations, the simple formula for calculating the new phase length is given. A predefined constant value is assigned to $ph_{requested}$ that denotes the number of phases requested during the whole search process. For this study, $ph_{requested} = 100$. Using it, a duration of a phase ($ph_{duration}$) is calculated by dividing the total execution time by $ph_{requested}$. The resulting value is divided by t_{subset} to reach the new pl' . If the calculated value is smaller than pl , then it is utilised as the new phase length. This process is repeated at the end of each phase.

$$\begin{aligned} ph_{duration} &= t_{total} / ph_{requested} \\ pl' &= ph_{duration} / t_{subset} \end{aligned} \quad (3)$$

The utilised constant values as well as the provided rules for adaptation purposes regarding the ADHS are set based on the idea of giving chance to the existing heuristics in the heuristic set to show their performance.

3.1.2 Learning automata

A learning automaton is a finite state machine that aims at learning the optimal action out of a set of actions ($A = \{a_1, \dots, a_n\}$) through interaction with an environment in a trial and error manner [33]. During a learning process, the environmental response ($\beta(t) \in [0, 1]$) to the selected action is used to update a probability vector p consisting of the selection probabilities of the actions. The update operation is carried out using an update scheme (U). The update scheme of the learning automata is based on equations (4) and (5). Equation (4) refers the update operation for the applied action and Equation (5) is used to update the probabilities of the rest of the actions.

$$\begin{aligned} p_i(t+1) &= p_i(t) + \lambda_1 \beta(t)(1 - p_i(t)) \\ &\quad - \lambda_2(1 - \beta(t))p_i(t) \end{aligned} \quad (4)$$

if a_i is the action taken at time step t

$$\begin{aligned} p_j(t+1) &= p_j(t) - \lambda_1 \beta(t)p_j(t) \\ &\quad + \lambda_2(1 - \beta(t))[(r-1)^{-1} - p_j(t)] \end{aligned} \quad (5)$$

if $a_j \neq a_i$

In [28], a learning automaton with a linear update scheme, *linear reward-inaction*, was employed as a heuristic selection mechanism. During the learning process, actions were considered as low-level heuristics. Heuristics that find new best solutions were rewarded. The changing heuristic probabilities were used like roulette wheel selection. In this study, *linear reward-punishment* scheme is used to update heuristic

probabilities with respect to finding a new best solution, improving the current solution, worsening the current solution and finding a solution with equal quality as the current solution. Related learning rates were set in a decreasing manner in the given order as $\{\lambda_1 = 0.1, \lambda_1 = 0.01, \lambda_2 = 0.002, \lambda_2 = 0.001\}$ without extensive tuning. The first two cases refer to the learning rates that are used for rewarding and the last two options show the learning rates for punishment. Differently from the above mentioned application of the learning automaton, it is just used to keep track of performance changes during different phases of the dynamic heuristic set process. In the beginning of each phase, the learning probabilities are reset. In addition, different learning rates are used for different heuristics. The underlying idea behind this approach is to project the speed of heuristics to the probability updates because it would be unfair to use the same reward/punishment for a very slow and a fast heuristic [4]. For instance, HyFlex contains local search heuristics that only return improving or equal solutions. They are generally expected to be more time consuming compared to other heuristic types. For that reason, the performance differences among heuristics should be interpreted based on their speed. The learning rate of the heuristics are determined using Equation (6). In this equation, C_{max} refers to the number of moves spent per unit time by the fastest heuristic. $C_{i,move}$ denotes the same value for heuristic i . The resulting value (λ_{mult}) is used as a multiplier for the initial learning rate. If this multiplier makes the related learning rate increase more than its predetermined limit, then the learning rate is set to a predetermined value.

$$\lambda_{mult} = C_{max}/C_{i,move} \quad (6)$$

3.1.3 Relay hybridisation

In [30], heuristics were divided into two types: mutational heuristics and hill climbers. They were used within four different frameworks. One of the frameworks, F_C , offers selecting a mutational heuristic first and applying a pre-determined hill climber to the visited solution by the selected mutational heuristic. The experimental results showed that F_C is a very effective strategy to use. In this study, we proposed a relay hybridisation approach to determine a pair of heuristics that can be used consecutively, like F_C , to find superior results. For that purpose, a list of the best heuristics, when applied just after a specific heuristic, is determined. This list is updated in *First-In-Last-Out* manner, by adding a new heuristic to the end of the list and removing the first one to keep the size of the heuristic set fixed if the list is full. After applying a relatively worse performing heuristic based on the provided performance metric, a heuristic is randomly chosen from its list and applied to the solution generated by this heuristic. Since the heuristic list can include different heuristics more than once, a random selection strategy chooses a heuristic among weighted heuristics. For instance, in a list size of 10, a specific heuristic may be present 5 times while the rest of the list may be occupied by the other heuristics. In such a list, the probability of selecting the 5-times occurring heuristic is 50%. That is, even if the selection mechanism is random, the end product is a weighted selection mechanism like roulette wheel.

This strategy is applied based on the value of (C_{phase}/pl) . C_{phase} denotes the number of iterations passed within a phase. A random probabilistic value $p \in [0 : 1]$ is generated and the relay hybridisation is applied if $p \leq (C_{phase}/pl)$. The pseudocode of the method is depicted in Algorithm 1.

Algorithm 1: Relay hybridisation

Input: $l_{size} > 0 \wedge p, p' \in [0 : 1]$
if $p \leq (C_{phase}/pl)$ **then**
 1 | select LLH_{tabu} and apply to $S \rightarrow S'$;
 2 | **if** $size(l_i) > 0$ and $p' \leq 0.5$ **then**
 3 | | select a LLH from l_i and apply to $S' \rightarrow S''$;
 4 | **else**
 5 | | select a $LLH_{nonTabu}$ and apply to $S' \rightarrow S''$;
 | **end**
end

In Figure 1, an example of the relay hybridisation is shown. In this example, applying heuristic pairs involving $LLH_2 \rightsquigarrow LLH_5$ and $LLH_2 \rightsquigarrow LLH_3$ found new best solutions before. Whenever LLH_2 is called and if the relay hybridisation is decided to be used, then it is determined whether a consecutive heuristic is randomly chosen from the heuristics that are currently available in the heuristic subset or from its pair list. The selected heuristic is consecutively applied to the solution at hand. If the resulting solution is a new best solution, then the selected heuristic is added to the end of the list.

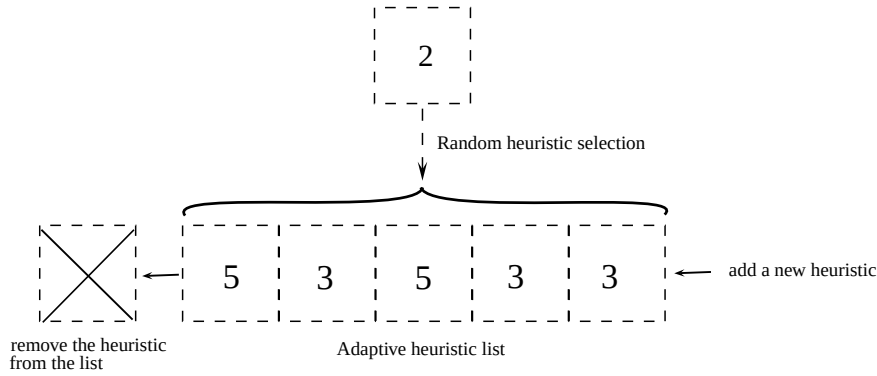


Fig. 1 Relay hybridisation

For some problem domains with some heuristic sets, it may not be possible to find effective heuristic pairs. Therefore, a similar tabu strategy that is used for the heuristic exclusion process is employed. If no new best solution is found by the relay hybridisation after a phase, then this feature is disabled for one phase. Furthermore, if this consecutively happens, then the tabu duration is increased by 1. This value stays the same whenever the tabu duration reaches its upper bound. If it can find a new best solution again, then the tabu duration is set to its initial value, namely 1.

Heuristic adaptation: HyFlex provides an opportunity to modify some of the heuristics in an informed manner. It is possible to increase or decrease the perturbation effect of a mutational heuristic. In addition to that, it is also allowed to change the depth of the search for local search heuristics. In the proposed approach, the deterioration

effects of the mutational heuristics and the search depth of the hill climbers was updated only for the relay hybridisation. If a hill climber can find a new best solution more than five times, then its search depth is set to 1.0. Otherwise, its value is set to $(0.1 + (ph_{passed}/pl)0.4)$. ph_{passed} is the number of iterations passed during the current phase. The second part is used also for the mutation operators.

3.2 Move acceptance

Move acceptance mechanisms have more influence on the performance of a hyper-heuristic [30]. They determine the way to traverse the search space. One of the main concerns of the move acceptance mechanisms is the acceptability of worsening solutions. Accepting a worsening solution is a widely accepted strategy to prevent from getting stuck around a solution. In [28], a move acceptance strategy was proposed, i.e. iteration limited threshold accepting (ILTA) which accepts worsening solutions in a controlled manner. ILTA immediately accepts improving or equal solutions. If the hyper-heuristic cannot find new best solutions during a pre-determined number of iterations, then, a worsening solution is accepted. This operation is decided based on the value of the best solution found and a constant value determining a range. In [27], an adaptive version of ILTA (AILTA) was proposed. In this move acceptance, a second and higher range value is determined. If the hyper-heuristic cannot find a new best solution using a given range value, then this range value is increased to enable accepting much worse solutions. This is required to get rid of local optima.

Algorithm 2: AILLA Move Acceptance

```

Input:  $i = 1 \wedge K \geq k \geq 0 \wedge l > 0$ 
for  $i=0$  to  $l-1$  do  $best_{list}(i) = f(S_{initial})$ 
1 if  $adapt\_iterations \geq K$  then
2   if  $i < l - 1$  then
3      $i++$ 
4   end
5   end
6 if  $f(S') < f(S)$  then
7    $S \leftarrow S'$ 
8    $w\_iterations = 0$ 
9   if  $f(S') < f(S_b)$  then
10     $S_b \leftarrow S'$ 
11     $i = 1$ 
12     $adapt\_iterations = 0$ 
13     $best_{list}.remove(last)$ 
14     $best_{list}.add(0, f(S_b))$ 
15  end
16 else if  $f(S') = f(S)$  then
17    $S \leftarrow S'$ 
18 else
19    $w\_iterations++$ 
20    $adapt\_iterations++$ 
21   if  $w\_iterations \geq k$  and  $f(S') \leq best_{list}(i)$  then
22      $S \leftarrow S'$  and  $w\_iterations = 0$ 
23   end
24 end

```

In the proposed hyper-heuristic, the underlying idea behind list-based threshold accepting [25] was employed to decrease the parameter dependency of AILTA. The new move acceptance strategy, i.e. adaptive iteration limited list-based threshold accepting (AILLA), accepts worsening solutions using the fitness values of the previously visited best solutions. The best fitness visited previously is used as the first threshold value. If it is not good enough to find a new best solution, then a higher fitness from the list ($best_{list}$ with l size) is used to decide about the acceptability of the worsening solutions. The pseudocode of AILLA is presented in Algorithm 2. In addition, the iteration limit (k) is updated whenever a new best solution is found. In Equation (7), the details of this update process is shown. In this equation, $iter_{elapsed}$ is the number of iterations elapsed and t_{exec} refers to the total execution time.

$$k = \begin{cases} ((l-1) \times k + iter_{elapsed})/l, & \text{if } cw = 0 \\ ((l-1) \times k + \sum_{i=0}^{cw} k \times 0.5^i \times tf)/l, & \text{otherwise} \end{cases} \quad (7)$$

$$tf = (t_{exec} - t_{elapsed})/t_{exec}$$

$$cw = \lfloor iter_{elapsed}/k \rfloor$$

4 Experiments

Experiments were carried out on the four problems within HyFlex, namely max SAT [21], bin packing [22], permutation flow shop [35] and personnel scheduling [17] problems. HyFlex provides 10 instances for each problem. Each instance was tested for 10 runs. Each run is executed for 10 minutes using Pentium Core 2 Duo 3 GHz PCs with 3.23 GB memory. In addition to the developed hyper-heuristic, three additional hyper-heuristics were tested. They are the combinations of simple random (SR) selection with AILLA, simulated annealing (SA) [3] and improving or equal (IE) move acceptance mechanisms.

4.1 Computational Results

For the bin packing problem, it is possible to determine an effective hyper-heuristic approach. The experimental results are shown in Table 2. Based on the Wilcoxon test, ADHS-AILLA performs significantly better than the other hyper-heuristics. When the heuristic selection mechanism ADHS is changed to SR, then a significant performance decrease comes out. This shows that for the bin packing problem with the given heuristic set, a learning or adaptation strategy concerning the selection procedure is required. On the other hand, there is no a statistically significant difference between SR-AILLA and SR-IE. However, they both generate superior results compared to SR-SA. This performance difference reveals the necessity of a more selective acceptance strategy regarding accepting worsening solutions. In other words, immediate acceptance of worsening solutions causes missing improving neighboring solutions.

For the permutation flowshop scheduling problem, again ADHS-AILLA statistically shows the best performance. Since SR-AILLA performs worse than ADHS-AILLA, it is apparent that ADHS provides a better hyper-heuristic opportunity. On the contrary, SR-AILLA consistently yields better results than SR-SA and SR-IE. This is a proof

BP	ADHS-AILLA				SR-AILLA			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	0.00594	0.00650	0.00703	0.00032	0.01177	0.01513	0.01668	0.00172
Inst2	0.00353	0.00652	0.00727	0.00107	0.01234	0.01583	0.02139	0.00262
Inst3	0.02029	0.02187	0.02275	0.00089	0.02110	0.02258	0.02371	0.00091
Inst4	0.01969	0.02138	0.02413	0.00151	0.02039	0.02239	0.02477	0.00138
Inst5	0.00034	0.00077	0.00456	0.00133	0.00034	0.00298	0.00510	0.00228
Inst6	0.00306	0.00314	0.00338	0.00010	0.00312	0.00440	0.00844	0.00214
Inst7	0.01099	0.01352	0.01584	0.00232	0.02226	0.02797	0.03223	0.00357
Inst8	0.02459	0.02716	0.03063	0.00245	0.08273	0.08684	0.09032	0.00269
Inst9	0.07273	0.07830	0.08560	0.00360	0.11710	0.12361	0.13272	0.00533
Inst10	0.01692	0.02292	0.02997	0.00377	0.03755	0.04219	0.04982	0.00449

BP	SR-SA				SR-IE			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	0.04048	0.04171	0.04441	0.00142	0.01197	0.01630	0.02041	0.00204
Inst2	0.03999	0.04213	0.04521	0.00179	0.01192	0.01521	0.01678	0.00197
Inst3	0.02455	0.02618	0.02723	0.00088	0.02275	0.02403	0.02519	0.00104
Inst4	0.02686	0.02906	0.03291	0.00174	0.02347	0.02494	0.02651	0.00111
Inst5	0.02230	0.02592	0.02773	0.00196	0.00035	0.00508	0.00647	0.00173
Inst6	0.02500	0.02536	0.02592	0.00034	0.00325	0.00555	0.00844	0.00250
Inst7	0.11090	0.11628	0.12430	0.00441	0.02217	0.02738	0.03205	0.00321
Inst8	0.12023	0.12507	0.13022	0.00305	0.08070	0.08456	0.08816	0.00282
Inst9	0.11598	0.11973	0.12552	0.00346	0.11372	0.12454	0.13352	0.00584
Inst10	0.03335	0.03451	0.03620	0.00089	0.03738	0.04309	0.05192	0.00559

Table 2 The results of the hyper-heuristics on the bin packing problem

FS	ADHS-AILLA				SR-AILLA			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	6263	6293	6308	12,9	6285	6299	6325	10,66
Inst2	6251	6273	6309	18,56	6226	6258	6280	17,57
Inst3	6315	6337	6354	11,41	6326	6339	6365	12,58
Inst4	6307	6358	6366	18,16	6323	6348	6366	20,14
Inst5	6365	6383	6402	12,43	6355	6389	6412	16,97
Inst6	10494	10499	10506	3,36	10502	10511	10526	8,53
Inst7	10923	10928	10972	15,5	10923	10934	10958	14,49
Inst8	26261	26301	26345	30,38	26301	26384	26435	44,73
Inst9	26754	26792	26816	21,75	26809	26865	26923	38,49
Inst10	26618	26676	26710	24,92	26638	26714	26751	33,98

FS	SR-SA				SR-IE			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	6291	6326	6364	20,12	6321	6347	6388	22,26
Inst2	6249	6296	6322	24,62	6261	6303	6340	26,68
Inst3	6335	6359	6386	18,17	6324	6369	6428	27,29
Inst4	6323	6351	6366	17,73	6330	6357	6374	15,24
Inst5	6364	6406	6456	27,56	6374	6405	6446	19,17
Inst6	10501	10538	10556	19,46	10497	10533	10555	19,9
Inst7	10923	10947	11017	30,33	10923	10947	11017	26,55
Inst8	26485	26519	26564	32,24	26413	26500	26649	74,37
Inst9	26888	26964	27046	42,16	26881	26978	27093	70,01
Inst10	26736	26781	26890	45,14	26732	26806	26876	59,29

Table 3 The results of the hyper-heuristics on the permutation flowshop scheduling problem

for the effect of the proposed move acceptance strategy. Besides, the performances of SR-SA and SR-IE are similar.

The experimental results (Table 4) on the personnel scheduling problem are hard to interpret because the number of iterations spent during the given execution time is limited. That is, there is not much time for exploration. Thus, a hyper-heuristic that reacts fast may have a higher chance to provide the best solutions. With the given

PS	ADHS-AILLA				SR-AILLA			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	3300	3326	3388	27.46	3317	3352	3391	22.93
Inst2	2115	2231	2495	108.26	2070	2362	2592	151.26
Inst3	295	338	380	25.41	405	1039	1885	677.92
Inst4	10	21	30	5.27	22	29	35	4.2
Inst5	17	23	29	3.88	26	30	36	3.06
Inst6	19	25	30	3.33	19	32	38	5.47
Inst7	1106	1159	1216	50.19	1117	1240	1404	78.35
Inst8	2176	2268	2359	56.46	2179	2255	2369	59.76
Inst9	3130	3252	3435	102.2	3166	3316	3527	118.08
Inst10	9438	9624	9781	107.33	9526	9642	9826	103.53

PS	SR-SA				SR-IE			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	3313	3368	3426	37.22	3307	3355	3432	36.6
Inst2	2295	2556	2725	135.63	2158	2558	3430	348.21
Inst3	480	1560	3160	1047.48	435	1033	1870	683.33
Inst4	17	27	36	5.1	25	30	36	3.44
Inst5	21	29	38	5.02	23	30	38	5.25
Inst6	21	31	43	6.33	24	32	39	5.44
Inst7	1100	1212	1411	102.46	1116	1268	1510	130.83
Inst8	2186	2256	2334	36.83	2183	2266	2464	73.47
Inst9	3159	3289	3347	61.69	3164	3296	3442	87.98
Inst10	9490	9646	9824	101.65	9380	9669	10050	192.04

Table 4 The results of the hyper-heuristics on the personnel scheduling problem

SAT	ADHS-AILLA				SR-AILLA			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	8	15	20	4.65	8	11	17	2.73
Inst2	24	29	37	4.62	20	25	29	2.82
Inst3	23	25	30	2.55	17	21	29	3.27
Inst4	7	9	11	1.51	7	10	13	1.84
Inst5	2	6	10	2.76	3	4	6	1.16
Inst6	3	6	11	2.25	3	5	7	1.45
Inst7	9	15	23	4.6	12	15	19	2.67
Inst8	8	18	32	8.17	8	24	39	11.64
Inst9	2	5	9	2.2	1	3	6	1.49
Inst10	2	5	8	1.89	2	3	6	1.35

SAT	SR-SA				SR-IE			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	6	9	14	2.26	11	14	21	3.27
Inst2	20	24	27	2.21	24	27	31	2.07
Inst3	16	18	22	1.63	19	23	28	2.75
Inst4	5	6	7	0.82	18	24	27	2.95
Inst5	1	3	4	0.97	5	10	12	2.41
Inst6	1	3	4	1.14	6	11	14	2.46
Inst7	9	12	15	2.23	26	30	36	3.34
Inst8	3	9	16	4.04	37	44	50	3.84
Inst9	1	3	5	1.16	3	5	8	1.73
Inst10	0	2	3	0.82	3	5	7	1.51

Table 5 The results of the hyper-heuristics on the Max SAT problem

learning strategy, ADHS-AILLA performs the best. There is no statistically different performance between the rest of the hyper-heuristics.

In Table 5, the results on Max SAT are given. For this problem, SR-SA provides significant performance improvement differently from the other three problems. Besides, SR-AILLA takes the second place and leads to superior results compared to ADHS-AILLA and SR-IE. These results show that ADHS worsens the performance of

the hyper-heuristic for Max SAT. Moreover, SA functions way better than AILLA and IE.

	BP	MSAT	FS	PS	OverAll
ADHS-AILLA	85.0	80.0	87.0	56.0	308.0
HH1	38.0	37.0	27.5	63.5	166.0
HH2	39.0	70.5	24.0	63	196.5
HH3	76.0	47.5	19.5	19	162.0
HH4	59.0	34.5	61.0	54	208.5
HH5	9.0	0.5	10.0	37.0	56.5
HH6	61.0	67.0	71.0	1.0	200.0
HH7	22.0	42.0	26.0	58.0	148.0
HH8	1.0	11.0	64.0	38.5	114.5

Table 6 The comparison of ADHS-AILLA and the hyper-heuristics (HH1~HH8) from the CHeSC competition website (higher values are better)

In Table 6, ADHS-AILLA was compared to the hyper-heuristic results announced at the CHeSC 2011 competition website¹. The comparison was made via the Formula 1 scoring system. The best hyper-heuristic gets 10, the rest get 8,6,5,4,3,2,1 based on their performance order. That is, only top eight hyper-heuristics earn points. The comparisons are carried out with only one run per instance. The comparison results show that ADHS-AILLA wins bin packing, Max SAT and the flowshop scheduling tracks. It comes fourth for the personnel scheduling problem. According to the over all performance, ADHS-AILLA wins.

4.2 Relay hybridisation

In Figure 2, the effect of the proposed relay hybridisation on different problems is shown. Consecutively applied heuristics reach new best solutions during the whole search process for the bin packing problem. On the other hand, for the max SAT and flowshop scheduling problems, this happens only at the early stages of the search. For personnel scheduling, the effect of the hybridisation is also limited. However, as was mentioned before, the number of iterations spent for this problem is quite small compared to the other problems. For the bin packing problem, mostly the crossover operator is useful for generating the first solution that is sent to the second heuristic. As the second heuristic, one of the local search heuristics (LLH_4) performs as the best second heuristic within pairs. One can consider LLH_0 , LLH_2 and LLH_3 as the other second heuristics. For max SAT, mutational heuristics (LLH_3) LLH_4 , with the following local search heuristics LLH_0 and LLH_1 help the hyper-heuristic for finding new best solutions. For the permutation flowshop, mutational (LLH_0 , LLH_1) and ruin-recreate (LLH_6 , LLH_6) heuristics with local search heuristics (LLH_7 , LLH_8 , LLH_{10}) construct effective heuristic pairs. For personnel scheduling, again mutational (LLH_{11}) and ruin-recreate heuristics (LLH_5 , LLH_6) with one particular local search heuristic (LLH_3) perform effectively. All these heuristic combinations indicate that the proposed hybridisation mechanism can determine effective heuristic pairs. As expected, heuristic pairs mostly behave as memetic algorithms.

¹ <http://www.asap.cs.nott.ac.uk/chesc2011/> – results on 15/01/2011

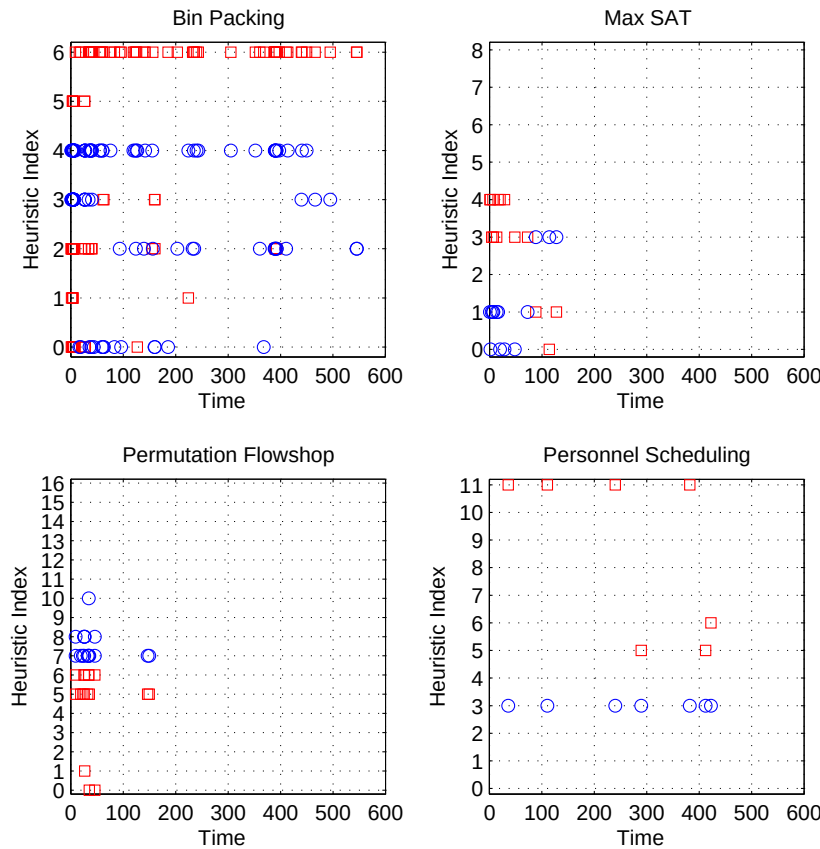


Fig. 2 Heuristic combinations from relay hybridisation (squares and circles show the applied heuristics consecutively)

4.3 Heuristic selection

In Figure 3, the effect of the proposed heuristic selection mechanism is shown. For the bin packing and permutation flowshop scheduling problems, ADHS calls more heuristics than SR during the given execution time. This situation reverses for the max SAT and personnel scheduling problems. This shows that the ADHS selection mechanism may invest time in time consuming but effective heuristics if it is required. In Figure 4, the number of heuristic calls of each heuristic over time is illustrated. For the given heuristic sets, the proposed selection strategy determines a distinction between heuristic performances. For the personnel scheduling and permutation flowshop scheduling problems, performance differences look stable. However, for the bin packing and max SAT problems, some heuristics behave differently during the search. In other words, some heuristics started to be selected more and others less. This shows that, for solving the bin packing and max SAT problems, considering performance changes compared to each other over time can be effective.

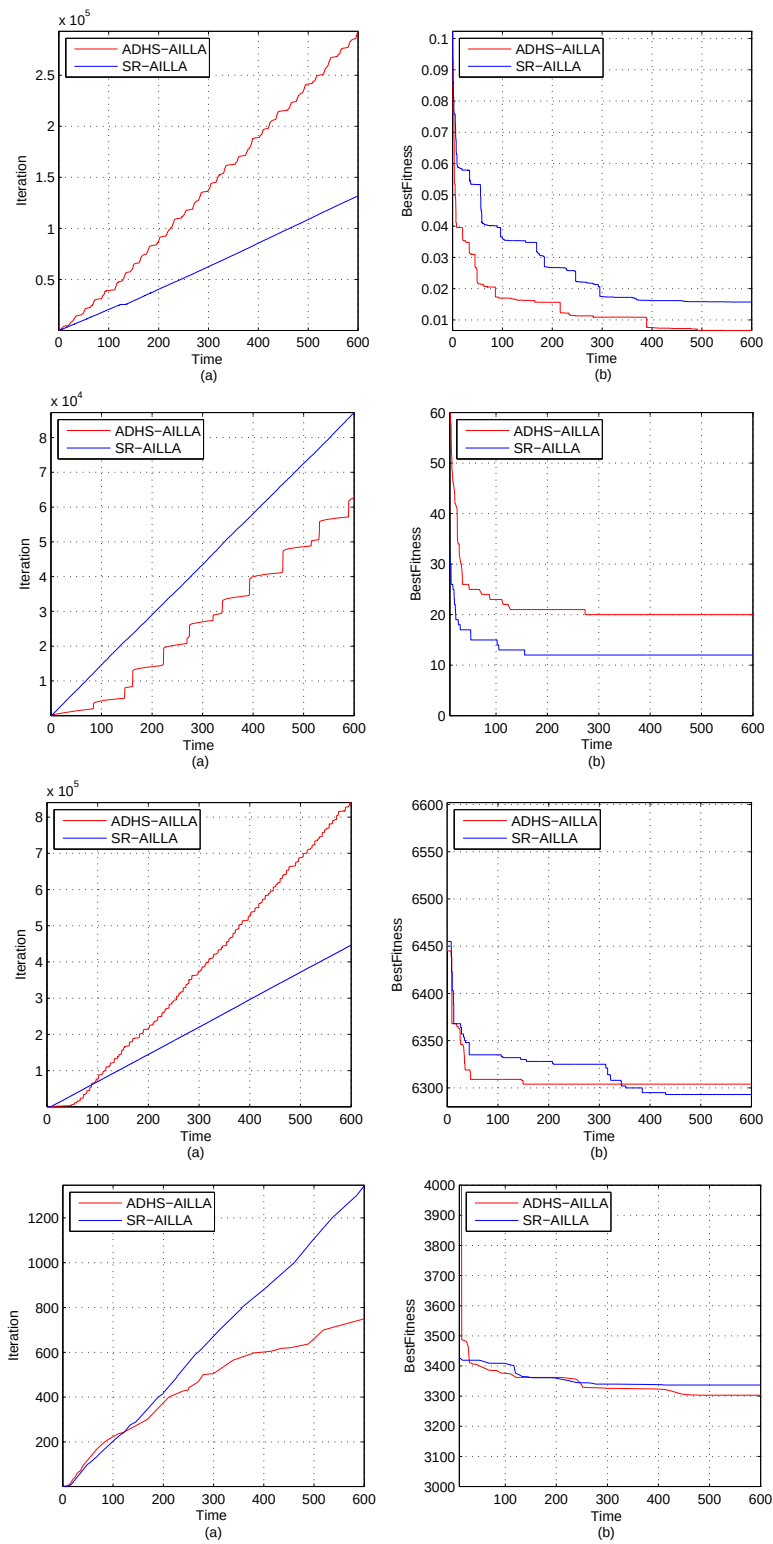


Fig. 3 The effect of the proposed selection mechanism (the chart pairs belong to the bin packing, max SAT, flowshop scheduling and personnel scheduling problems from top to bottom)

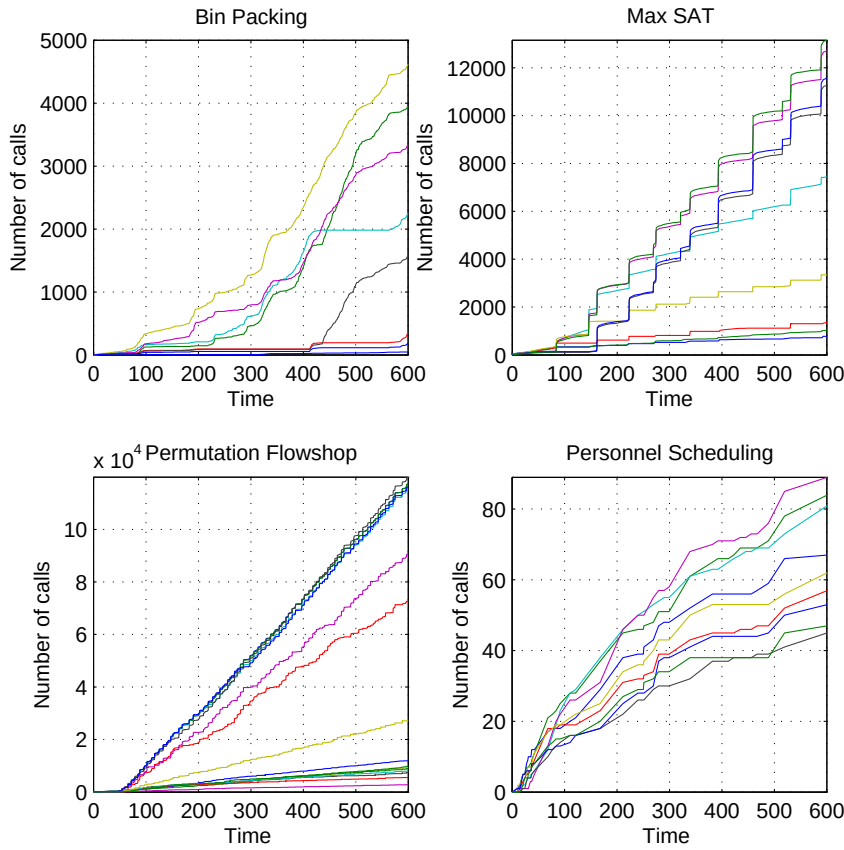


Fig. 4 The number of calls for each heuristic over the given problem domains by ADHS-AILLA

4.4 Varying performance of the heuristics

In Figure 5, the varying QI values of each heuristic during the search process are illustrated. The charts show that even if LLH_3 is a hill climber, it is frequently excluded from the heuristic set with the only crossover operator, LLH_6 . For the remaining heuristics, it is possible to see performance changes based on the employed performance metric. During early iterations, a mutational heuristic (LLH_0), a ruin-recreate heuristic (LLH_2) and a hill climber (LLH_4) take the lead to improve the solution at hand. However, after a while, the other ruin-recreate heuristic (LLH_1) with a mutational heuristic (LLH_5) start to perform better. These results show that the performance variation of different heuristics should be regularly monitored and analysed.

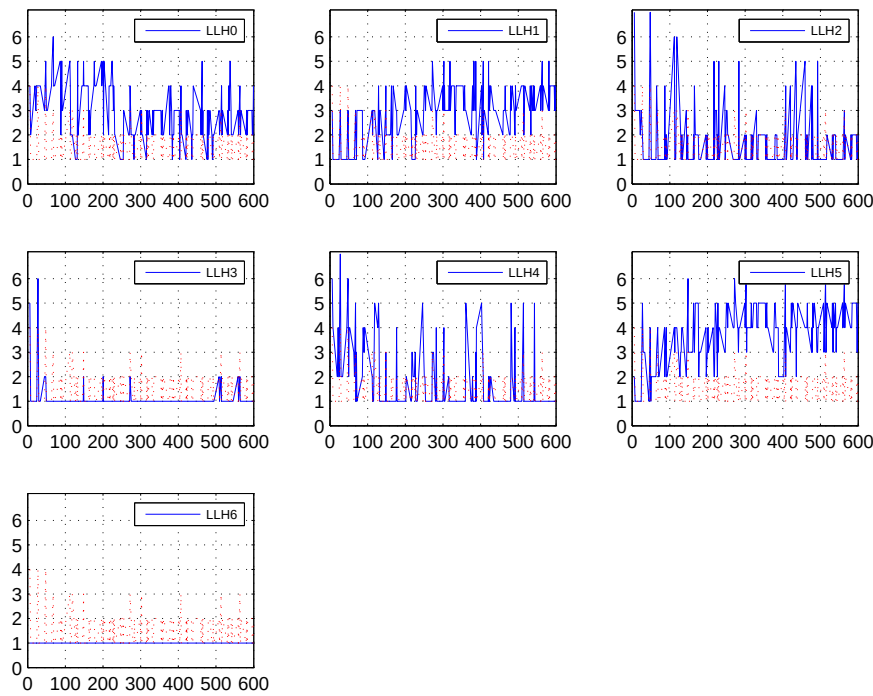


Fig. 5 QI -avg values of the bin packing heuristics (Quality index (QI) values regarding the performance of the heuristics are shown as solid lines, average (avg) of the QI values are shown as dotted lines. The heuristics with a lower QI than avg are excluded.)

4.5 Set size

In Figure 6, changes of the heuristic set size over time is demonstrated. For the bin packing problem, the exclusion process eliminates at most 3 heuristics and the set size generally changes between 4 and 5. For max SAT, fluctuations of the set size is considerably less. It can be explained by the number of phases because the existing SAT heuristics can spend more time than bin packing heuristics. The differences on the number of iterations per time is shown in Figure 3. For the flowshop scheduling problem, the set size is reduced to 6 out of 16 heuristics and it changes mainly between 6 and 9. For the last problem domain, even one phase could not be completed due to the previously mentioned speed issues.

5 Conclusion

In the literature, there exist different kinds of problems with various solution strategies. Many of these problems are hard to solve. Hence, there is a tendency of dedicated developing heuristic methods for solving these problems. Among these approaches, it is expected that some of them work exceptionally well on some problems or problem instances, but generate inferior results on others. Hyper-heuristics search over such

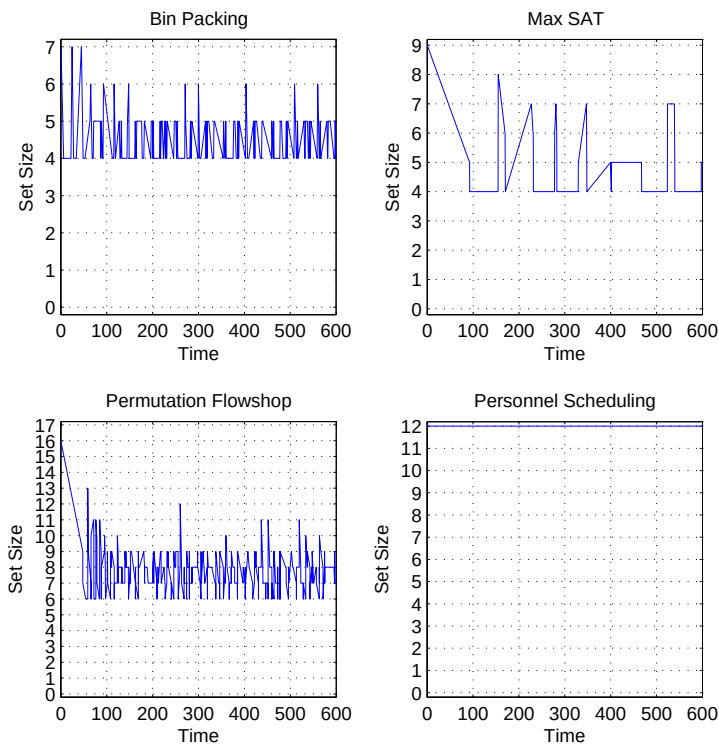


Fig. 6 Changes on the heuristic set size over time by ADHS-AILLA

low-level algorithms to provide generality by using their strengths. With problem-independency, their generality level increases. In this research, a new selection hyper-heuristic was designed and implemented on the hyper-heuristic software framework, i.e. HyFlex. The developed approach consists of a dynamic heuristic set strategy that determines the best heuristic subsets along a number of iterations. For increasing the effectiveness of this elimination method, a learning automaton and a pairwise heuristic hybridisation mechanism were employed. Also, a new threshold based dynamic move acceptance strategy was accommodated. A set of experiments was carried out with three additional hyper-heuristics over four problem domains with their specific heuristic sets. The experimental results indicated that the designed hyper-heuristic is an effective strategy for the given problem instances and the heuristic sets. Although the hyper-heuristic performs well on average, it still has some issues. For the Max SAT problem, SA generates superior results compared to AILLA. Moreover, ADHS performs worse than SR for the same problem. The related results should be analysed to increase the performance of the proposed hyper-heuristic. In addition, since there is room for improvement concerning the personnel scheduling problem, the missing elements or misleading algorithm settings should be determined.

In the future, the mentioned drawbacks will be investigated. Then, a feedback mechanism will be settled between the selection process and the move acceptance part. In addition, a two-phase heuristic selection mechanism consisting of selecting a

heuristic subset and choosing a heuristic from the selected heuristic set in connection with a feedback mechanism will be utilised. Furthermore, the adaptive characteristic of the move acceptance strategy will be boosted.

References

1. E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.
2. M. Bader-El-Den, R. Poli, and S. Fatima. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, 1(3):205–219, 2009.
3. B. Bilgin, P. Demeester, M. Misir, W. Vancroonenburg, G. Vanden Berghe, and T. Wauters. A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition. In *the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10) - the Nurse Rostering Competition*, 2010.
4. M. Bowling and M. Veloso. Rational and convergent learning in stochastic games. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 1021–1026. Citeseer, 2001.
5. E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942–958, 2010.
6. E.K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, and J.A. Vazquez-Rodriguez. Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'09)*, pages 790–797, Dublin, Ireland, August 10–12 2009.
7. E.K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. *Handbook of Meta-Heuristics*, chapter Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pages 457–474. Kluwer Academic Publishers, 2003.
8. E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J.R. Woodward. A classification of hyper-heuristic approaches. *Handbook of Metaheuristics*, pages 449–468, 2010.
9. E.K. Burke, M. Hyde, G. Kendall, and John Woodward. Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'07)*, pages 1559–1565, London, England, July 12–16 2007.
10. E.K. Burke, M.R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In T.P. Runarsson, H-G.B., E. Burke, J.J. Merelo-Guervos, L.D. Whitley, and X. Yao, editors, *Proceedings of the 9th Parallel Problem Solving from Nature (PPSN'IX)*, volume 4193 of LNCS, pages 860–869, Reykjavik, Iceland, September 9–13 2006. Springer-Verlag.
11. E.K. Burke, G. Kendall, M. Misir, and E. Ozcan. Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, pages 1–18, 2010. 10.1007/s10479-010-0782-2.
12. E.K. Burke, G. Kendall, D.L. Silva, R. O'Brien, and E. Soubeiga. An ant algorithm hyper-heuristic for the project presentation scheduling problem. In *Proceedings of the Congress on Evolutionary Computation 2005 (CEC'05). Volume 3*, pages 2263–2270, 2005.
13. E.K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics*, 9(3):451–470, 2003.
14. E.K. Burke, S. Petrovic, and R. Qu. Case based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132, 2006.
15. K. Chakhlevitch and P. Cowling. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In G. Raidl and J. Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3448 of LNCS, pages 23–33. Springer Berlin / Heidelberg, 2005.
16. P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *PATAT '00: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, pages 176–190, London, UK, 2001. Springer-Verlag.
17. T. Curtois, G. Ochoa, M. Hyde, and J. A. Vazquez-Rodriguez. A hyflex module for the personnel scheduling problem. Cs technical report, Univeristy of Nottingham, 2010.

18. K.A. Dowsland, E. Soubeiga, and E.K. Burke. A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research*, 179(3):759–774, 2007.
19. A.S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):31–61, 2008.
20. L. Han and G. Kendall. An investigation of a tabu assisted hyper-heuristic genetic algorithm. In *Proceedings of Congress on Evolutionary Computation (CEC'03)*, volume 3, pages 2230–2237, 2003.
21. M. Hyde, G. Ochoa, T. Curtois, and J. A. Vazquez-Rodriguez. A hyflex module for the maximum satisfiability (max-sat) problem. Cs technical report, Univeristy of Nottingham, 2010.
22. M. Hyde, G. Ochoa, T. Curtois, and J. A. Vazquez-Rodriguez. A hyflex module for the one dimensional bin packing problem. Cs technical report, Univeristy of Nottingham, 2010.
23. G. Kendall and N.M. Hussin. An investigation of a tabu-search-based hyper-heuristic for examination timetabling. In *Multidisciplinary scheduling: theory and applications: 1st International Conference, MISTA'03: Nottingham, UK, 13-15 August 2003: selected papers*, page 309. Springer Verlag, 2005.
24. G. Kendall and N.M. Hussin. A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology. In *Proceedings of the 5th Practice and Theory of Automated Timetabling (PATAT'04)*, volume 3616 of *LNCS*, pages 270–293. Springer, 2005.
25. D.S. Lee, V.S. Vassiliadis, and J.M. Park. List-based threshold-accepting algorithm for zero-wait scheduling of multiproduct batch plants. *Industrial & Engineering Chemistry Research*, 41(25):6579–6588, 2002.
26. J.G. Marn-Blázquez and S. Schulenburg. A hyper-heuristic framework with xcs: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In T. Kovacs, X. Llor, K. Takadama, P.L. Lanzi, W. Stolzmann, and S.W. Wilson, editors, *IWLCS*, volume 4399 of *LNCS*, pages 193–218. Springer, 2007.
27. M. Misir, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'10)*, pages 2875–2882, Barcelona, Spain, July 18–23 2010.
28. M. Misir, T. Wauters, K. Verbeeck, and G. Vanden Berghe. A new learning hyper-heuristic for the traveling tournament problem. In *Proceedings of the 8th Metaheuristic International Conference (MIC'09)*, Hamburg, Germany, 2009.
29. A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer Decision-Making*, pages 523–544. Kluwer Academic Publishers, 2003.
30. E. Ozcan, B. Bilgin, and E.E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.
31. E. Ozcan, M. Misir, G. Ochoa, and E.K. Burke. A reinforcement learning - great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing*, 1(1):39–59, 2010.
32. P. Ross, S. Schulenburg, J.G. Marín-Blázquez, and E. Hart. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02)*, pages 942–948, New York, July 9–13 2002. Morgan Kaufmann Publishers.
33. M.A.L. Thathachar and P.S. Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers, 2004.
34. W. Vancroonenburg, M. Misir, B. Bilgin, P. Demeester, and G. Vanden Berghe. A hyper-heuristic approach for assigning patients to hospital rooms. In *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10)*, Belfast, Northern Ireland, August 10–13 2010.
35. J. A. Vazquez-Rodriguez, G. Ochoa, T. Curtois, and M. Hyde. A hyflex module for the permutation flow shop problem. Cs technical report, Univeristy of Nottingham, 2010.

Robust Target Achievement for Decision-Making with Application to Workforce-Inventory Planning

Tsan Sheng Adam Ng · Charlle Lee Sy

Abstract In this work, we propose a decision-making approach that evaluates the quality of a solution's ability to achieve targets under uncertainty. To this end, we develop a mathematical programming model that is computationally efficient and effective in hedging against uncertainties when limited information on the uncertainties are known. Our research synthesizes recent developments in robust optimization technology and the target-oriented behavior of decision agents. A numerical case study involving a workforce-inventory planning system demonstrates significant improvements of the proposed solution in achieving targets under uncertainty.

1 Introduction

Decision-making under uncertainty is one of the most important and challenging areas in operations research. Stochastic optimization approaches, in particular expected utility maximization, are the common standard de facto for problem modeling amongst researchers and practitioners.

On the other hand, it is well-acknowledged in decision theory that real world agents rarely 'optimize' their decisions. Doing so implies a need for a perfect model of the system [20], which is highly improbable in reality. Rather than formulating and solving complicated optimization problems, real-world agents often simply choose the first available actions which ensure that certain desired targets would be achieved. This is the basic premise of the *satisficing approach* that is based on the postulation of bounded rationality of human decision-makers. For instance, in the decision-making process of a firm, it is to be expected that the firm's decision would involve the attainment of a certain level of profit rather than the maximization of profit[19].

The adaptation of the satisficing approach in optimization problems was pioneered by [7]. Rather than expected utility maximization, the authors defined a satisficing

Tsan Sheng Adam Ng
Department of Industrial and Systems Engineering, National University of Singapore
E-mail: isentsa@nus.edu.sg

Charlle Lee Sy
Department of Industrial and Systems Engineering, National University of Singapore
E-mail: charlle_sy@nus.edu.sg

solution as one that has the highest probability of achieving the desired target levels. More recently, [6] and [4] showed that maximizing expected utility is mathematically equivalent to maximizing the probability that the uncertain consequences of a decision are preferable to an uncertain benchmark.

However, a major drawback of the above approaches is that the knowledge of the exact probability distributions governing the uncertainties can be a very strong assumption. Failure to satisfy these assumptions can severely compromise the validity of the analysis. Furthermore, even if the distributional assumptions are reasonable, probabilistic optimization models are generally highly intractable. In this work, we propose a satisficing approach to decision-making under uncertainty that avoids the need to fully specify the distributions of the uncertainties. In particular, we propose a satisficing measure that evaluates the level of target achievement of a decision under uncertainty. A key advantage of our approach is that the resulting decision problem is an analytically tractable optimization model that maximizes a system's tolerance to uncertainty.

The rest of the paper is organized as follows. In Section 2, we discuss the robust optimization approach of handling data uncertainties. The essential background on the aspiration level approach in decision-making that is utilized in this work is then provided. Section 3 introduces the main technical components of our proposed robust target-achievement approach. These include the development of a satisficing measure based on an axiomatic framework and its integration to an optimization model. Section 4 presents a numerical study on the application of the proposed approach using a workforce-inventory planning system. Finally, Section 5 concludes our work.

2 Literature Review

One notion of introducing robustness in decision-making is to immunize optimization models against data uncertainty ([1] and [2]). In these works, the goal is to build a computationally tractable robust counterpart which produces solutions that remain feasible under variations of data within assumed bounds or uncertainty sets. To allow flexibility, such models enable a decision-maker to trade-off the level of conservativeness and the quality of the solution, justified through probability upper bounds of constraint violations when the size of the uncertainty sets change (see for example [3]). More recently, robust optimization technology has also incorporated partial information of the underlying probability distributions, such as moments information (mean, support, variance) to generate distributionally robust solutions [12]. That is, solutions that guarantee good performance bounds over the family of distributions characterized by the given partial information.

The use of partial information addresses the drawbacks identified in probabilistic approaches. As mentioned in §1, it is difficult to evaluate the probability of target achievement. Probability functions also lack quasi-concavity properties. These prevent them to guarantee convex preferences, thereby leading to computational intractability. Furthermore, maximizing the success probability assumes that the decision agent is indifferent to the level of shortfall. This can be particularly problematic and undesirable in the face of catastrophic events of small probability but severe associated shortfalls. Indeed, studies have suggested that subjects are not insensitive to these down-side risks; we refer interested readers to [17] and [9] for more detailed treatments of these aspects.

Motivated by the issues associated with probability optimization as discussed, [8] proposed a class of satisficing measures based on an axiomatic framework. An example of which includes the shortfall aspiration level, which is a tractable alternative to probability measures. This measures the success in achieving the target and the expected level of under-performance or shortfall. In this work, we introduce a satisficing measure based on this framework that seeks to maximize the parameter variation ranges which the system can tolerate. Consequently, the system is able to achieve its design targets robustly under any realization of the uncertainties within the supports of these parameter variation ranges.

3 Robust Target Achievement Approach

This section presents our proposed methodology of a robust target achievement using satisficing measures. We use the axiomatic definition of satisficing measures proposed by [5]. This is stated in the following.

Suppose we have the uncertain outcomes, v , that lie within the set of outcomes $\mathcal{V}$ (i.e. $v \in \mathcal{V}$). Let the target be denoted by τ such that the excess (shortage) of the outcomes with respect to τ is then $v - \tau$. For notational convenience, we denote v as the uncertain outcome normalized by the targets, i.e. $\tau = 0$. Thus, a satisficing solution achieves the target if and only if $v \geq 0$. The axiomatic definition of a satisficing measure ρ is then:

Definition 1: $\rho : \mathcal{X} \mapsto [0, 1]$, is a satisficing measure if it adheres to the following axioms:

1. Attainment content: If $v \geq 0$, then $\rho(v) \geq 0$.
2. Non-attainment apathy: If $v < 0$, then $\rho(v) = 0$.
3. Monotonicity: If $v \geq u$, then $\rho(v) \geq \rho(u)$.
4. Gain continuity: $\lim_{\alpha \downarrow 0} \rho(v + \alpha) = \rho(v)$.

Attainment content implies that if the target is always achieved, then the satisficing level ρ attains its maximum (i.e. $\rho = 1$). On the other hand, *Non-attainment apathy* indicates that if the current position never achieves the target (i.e. under any realization), then it is always least preferable (i.e. $\rho = 0$). *Monotonicity* states that if an outcome v (state-wise) dominates outcome u , then the former is no less preferable $\rho(\tilde{v}) \geq \rho(\tilde{u})$. This indicates that outcome v achieves the target at least as well as outcome u . Finally, *Gain continuity* is a technical requirement meaning that if an infinitesimally small increment is made to the current position, the satisficing level would not change. The opposite however is not necessarily true. We introduce a satisficing measure that is consistent with Definition 1 in the succeeding subsections.

3.1 Modeling Data Uncertainties

We first describe the uncertainty set in the model by defining a vector of uncertain parameters $\tilde{z} = [\tilde{z}_1, \tilde{z}_2, \dots, \tilde{z}_L]$ as follows:

$$\tilde{z} = \bar{z} + z \quad (1)$$

In (1), $\bar{z}$ denotes the nominal values of the parameters and the perturbations z are such that

$$z \in [-\theta\bar{z}, \theta\bar{z}]$$

with the scalar parameter $\theta \in [0, 1]$ and $\bar{z} \geq 0$.

Hence, $\theta\bar{z}$ is the maximum possible perturbations in the set. In many practical problems, there are also additional information about the relationships between the uncertain parameters. Such information can help to reduce the uncertain outcome space significantly and hence should be incorporated when available. We use the below set of linear constraints as first-order approximations of such relationships,

$$Rz \leq e \quad (2)$$

where the $W \times L$ matrix R and the vector $e = [e_1, \dots, e_W]$ are estimated coefficients of the W relationships. Combining the above, for a given θ , we define the set of possible outcomes $F(\theta)$ as:

$$F(\theta) = \left\{ z \in \mathbb{R}^L : z \in [-\theta\bar{z}, \theta\bar{z}], Rz \leq e \right\}$$

Hence, $F(\theta)$ is a polyhedron parameterized by θ . It is clear that $F(\theta') \subseteq F(\theta)$ whenever $\theta \geq \theta'$. This formulation provides us with a functional means to evaluate the robustness of a solution. In particular, we say that a candidate solution has a robust target achievement of level θ if the system is able to meet all the targets for any $z \in F(\theta)$.

3.2 Robust Target Achievement Model

Our problem is then to maximize the robust target achievement of level θ through the calibration of the decision variables. That is, we seek a solution so that the targets are always achieved under the uncertain parameters arising from the set $F(\theta)$ for the largest value of θ .

There are also often constraints and requirements imposed on the decision space. These can include bounds on decision variable ranges or more general resource budgets that limit the total resource consumption associated with changing the settings of the variables. We assume that the feasible space of solution alternatives is modeled as the polyhedron $\mathcal{X}$, so that any $x \in \mathcal{X}$ is a feasible decision variable setting. The problem is then stated mathematically in the following formulation.

Problem $\mathcal{RTA}$

$$\begin{aligned} & \max_{x \in \mathcal{X}} \theta \\ & A \cdot x + B \cdot z \leq \tau \quad \forall z \in F(\theta) \end{aligned} \quad (3)$$

where $x = [x_1, x_2, \dots, x_M]$ is a vector of decision variables, $\tau = [\tau_1, \tau_2, \dots, \tau_N]$ is a vector of system targets and A and B are matrices of coefficients corresponding to x and z , respectively.

The set of constraints (3) must hold for all $z \in F(\theta)$. In order to make $\mathcal{RTA}$ amenable for solution using linear programming techniques, we apply the following Proposition to develop an equivalent formulation of problem $\mathcal{RTA}$ defined as the problem $\mathcal{RC}$.

Proposition 1: Define $P, Q \in \mathbb{R}^{N \times L}$ and $S \in \mathbb{R}^{N \times W}$ as matrices of variables. Problem $\mathcal{RTA}$ is then equivalent to the following:

Problem $\mathcal{RC}$

$$\max_{\mathbf{x} \in \mathcal{X}} \theta$$

s. t.

$$A \cdot \mathbf{x} + (P + Q) \cdot \theta \bar{\mathbf{z}} + S \cdot \mathbf{e} \leq \boldsymbol{\tau} \quad (4)$$

$$Q - P - S \cdot R \leq -B \quad (5)$$

$$P, Q, S \geq 0 \quad (6)$$

Proof: See Appendix A.

The formulation $\mathcal{RC}$ can be solved for the maximum θ^* by performing a line search in on $\theta \in [0, 1]$. That is, we identify an ascent direction that would improve the value of θ . For a fixed value of θ' , feasibility in (4) – (6) can be checked efficiently by solving a Phase 1 linear programming problem. If a feasible solution exists, we then have $\theta^* \geq \theta'$. Otherwise $\theta^* < \theta'$. The line search can then proceed by applying a linear or bisection search.

We justify our choice of maximizing θ by showing its relationship with the satisficing measures in Definition 1. First note that problem $\mathcal{RTA}$ can be re-written as:

$$\theta(\tilde{v}) = \begin{cases} \sup \{ \theta \in (0, 1) : \min_i \min_{\mathbf{z} \in F(\theta)} \{ -\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z} + \boldsymbol{\tau}_i \} \geq 0 \} & \text{if feasible,} \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

where $\tilde{v}$ is defined as

$$\tilde{v} = \min_{i=1, \dots, N} -\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z} + \boldsymbol{\tau}_i$$

and $\mathbf{A}_i, \mathbf{B}_i$ and $\boldsymbol{\tau}_i$ correspond to the i^{th} vectors of A, B and $\boldsymbol{\tau}$, respectively. That is, $\tilde{v}$ is the extreme-value random variable whose level is the worst achievement among the $i = 1, \dots, N$ design requirements. We then have the following technical result.

Proposition 2: $\theta(\tilde{v})$ is a satisficing measure in accordance to Definition 1 on the random variable $\tilde{v}$.

Proof: See Appendix B.

This result implies that by solving $\mathcal{RTA}$, we are indeed seeking a design solution $\mathbf{x}^*$ that maximizes the satisficing level as defined. In other words, $\mathbf{x}^*$ is undominated in the sense that it achieves all the system targets simultaneously under uncertainty at least as well as all other possible alternatives.

4 Workforce-Inventory Planning System

In this section, we demonstrate how the proposed approach can be implemented to improve the performance of a workforce-inventory planning system adapted from [18]. This system is illustrated in the stock and flow diagram in Figure 1, and is defined by the set of ordinary differential equations as in (8)-(11). For brevity, only the state equations are shown.

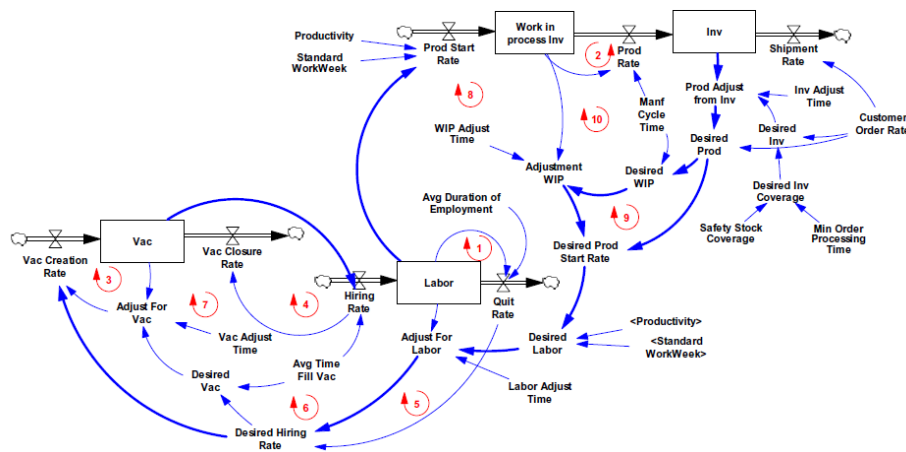


Fig. 1 Stock and flow diagram of a Workforce-Inventory Planning System [18]

$$INV(t) = PR - SR = \frac{WIP}{MCT} - COR \quad (8)$$

$$L\dot{A}B(t) = HR - QR = \frac{VAC}{ATF} - \frac{LAB}{ADE} \quad (9)$$

$$\begin{aligned} \dot{VAC}(t) = & VC_r R - VC_l R = \left(-\frac{VAT+ATF}{LAT \cdot VAT} + \frac{VAT+ATF}{VAT \cdot ADE} \right) LAB - \frac{VAT+ATF}{ATF \cdot VAT} VAC \\ & - \frac{ATF(WAT+MCT)+VAT(WAT+MCT)}{LAT \cdot PDY \cdot SWW \cdot VAT \cdot WAT \cdot IAT} INV - \frac{(ATF+VAT)WIP}{WAT \cdot LAT \cdot PDY \cdot SWW \cdot VAT} \end{aligned} \quad (10)$$

$$\dot{WIP}(t) = PSR - PR = LAB \cdot PDY \cdot SWW - \frac{WIP}{MCT} \quad (11)$$

where:

INV:	Inventory	WIP:	Work-in-process inventory
LAB:	Labor	VC_rR :	Vacancy creation rate
VAC:	Vacancy	VC_lR :	Vacancy closure rate
HR:	Hiring rate	PSR:	Production start rate
QR:	Quit rate	ADE:	Ave. duration of employment
COR:	Customer order	ATF:	Ave. time to fill vacancy
PDY:	Productivity	IAT:	Inventory adjustment time
SR:	Shipment rate	WAT:	WIP adjustment time
SWW:	Standard work week	MCT:	Manufacturing cycle time
PR:	Production rate	VAC:	Vacancy adjustment time
LAB:	Labor adjustment time		

In this system, inventory level (INV) is increased by the rate of production obtained from work-in-process (WIP) inventory and is depleted by the rate of product shipments (SR) determined by customer orders (COR). It is maintained at a desired level by means of adjusting the production rate (PR) and the workforce level (LAB). Specifically, these adjustments are made via changes in the production start rate (PSR),

which is determined by adjustments in the workforce and productivity (PDY). The quit rate (QR) triggers the creation of vacancies (VC_rR) that in turn leads to an increase in the hiring rate (HR). However, this hiring rate likewise affects the closure rate of vacancies (VC_lR) in the system.

The workforce-inventory system operates in an uncertain environment where variability in cycle times and other unforeseen scenarios may arise. The system is regarded as stable if, when perturbed from its steady operating point, it will return to it [15]. Stability is closely related to the system's ability to achieve and maintain intended performance under disturbances. For instance, the system's stability has direct implications on the ability of the management to control wide fluctuations on the inventory and workforce levels in the event of disruptions. These issues are of critical importance to stakeholders since they have direct impact on the costs associated with inventory and workforce.

In the current planning system, two lever parameters, namely the *Inventory Adjustment Time (IAT)* and the *Labor Adjustment Time (LAT)* are identified. The *IAT* refers to the production's responsiveness to shortfalls in inventory while *LAT* refers to the speed of the hiring process. The decision problem is then to determine the values of these lever parameters in order to achieve system stability under uncertainty.

4.1 Specification of Targets

In the literature (see [11],[13] and [14]), stability is characterized through a system's eigenvalue positions. This is operationalized by considering an eigenvalue (λ) to be of the form $\mu \pm i\omega$, where μ and ω are the real part and the imaginary part of the eigenvalue, respectively. The magnitude of μ is equivalent to the dampening rate of the state variables whereas ω determines the frequency of oscillations in the system. Furthermore, for a system to be stable, it is a necessary condition that the largest real part of the eigenvalue be negative (i.e. $\mu < 0$).

The eigenvalues can be computed through the linear time-invariant state space model of the system.¹ In the planning system, the eigenvalues corresponding to the same parameter values as defined by [18] are evaluated as:

$$\lambda_1 = -0.3531, \lambda_2 = -0.0095 + 0.0988i, \lambda_3 = -0.0095 - 0.0988i, \lambda_4 = -0.1380$$

The number of eigenvalues correspond to the number of state variables in the system. Clearly, the complex conjugate pairs λ_2 and λ_3 are the dominant eigenvalues since they have the largest real parts. They will dictate the overall behavior of the system. Therefore, we can deduce that the system is currently stable and is thus able to control the oscillations that may arise in this system. However, because these are relatively close to zero, the dampening of such oscillations would be very weak and it could take a significant amount of time for the system to achieve stability.

Furthermore, in the presence of uncertainty, the near-zero values of μ_2 and μ_3 can become problematic if it crosses the zero level to the positive, which would result in system instability. Specifically, it was observed that when the parameter values increase, the dominant eigenvalues became less negative. As an example, when the parameters take on the following values: $MCT = 10.5$, $VAT = 7$, $WAT = 8$, $LAT = 19$, $IAT = 12$, $ADE = 100$ and $ATFV = 8$, λ_2 and λ_3 become $0.0002 + 0.0901i$ and $0.0002 - 0.0901i$, respectively. This indicates an unstable system behavior. From these considerations,

¹ See Appendix C for the derivation of the state-space model.

we therefore want the real parts of μ_2 and μ_3 to become more negative. We can also decrease ω_2 and ω_3 in order to decrease the frequency of oscillations in this system. Table 1 shows target ranges that we assign for the eigenvalues in the formulation of our optimization model.

μ_2 and μ_3	ω_2 and ω_3	μ_1	μ_4
$[-0.03, -0.015]$	$[0.09, 0.10]$	$[-0.35, -0.25]$	$[-0.15, -0.09]$

Table 1 Target eigenvalues

4.2 Model Formulation

The influence of the decision variables and parameter uncertainties can be measured from eigenvalue sensitivity matrices.² The use of these matrices allows the stability problem to be modeled as in $\mathcal{RTA}$ with the set of eigenvalue ranges presented in Table 1 as the targets. We can also specify a set of decision and uncertain variables in the workforce-inventory system as in Table 2. For instance, to ensure that μ_2 and μ_3 stay within the identified target ranges, this requirement can be stated as in the form of (3):

$$\begin{aligned} A_{2,3} \cdot \mathbf{x} + B_{2,3} \cdot \mathbf{z} &\geq -0.03 \\ A_{2,3} \cdot \mathbf{x} + B_{2,3} \cdot \mathbf{z} &\leq -0.015 \end{aligned}$$

where $A_{2,3}$ and $B_{2,3}$ correspond to vectors in matrices A and B and are functions of the decision and uncertain variables, respectively.

It is to be noted that these constraints are also general enough to capture other requirements such as bounds on system state levels. For instance, we can constrain the state levels to fall within certain minimum and maximum values. As with the eigenvalue sensitivity matrix, we can define the A and B matrices as changes in the system output level with respect to the variables (see Appendix C).

Finally, as shown in Table 2, we also set bounds on the decision space by assuming the possible ranges IAT and LAT could have. Furthermore, the table also presents the uncertainty sets defined for the uncertain variables. These ranges indicate the degree of uncertainty assigned for the respective variables (i.e. possible change from the nominal values).

Decision variables	Bounds	Uncertain variables	Uncertainty set
IAT	5 to 30 weeks	ATF	-1.80 - 1.80
LAT	5 to 30 weeks	ADE	-22.60 - 22.60
		WAT	-1.40 - 1.40
		VAT	-0.90 - 0.90
		MCT	-1.80 - 1.80

Table 2 Decision and uncertain variables for workforce-inventory planning system

² The reader is referred to [10] for a detailed discussion of eigenvalue sensitivity matrices.

4.3 Results and Discussions

Using Proposition 1, the model $\mathcal{RTA}$ is transformed to its equivalent model $\mathcal{RC}$. It is then solved using the modeling toolbox ROME version 1.0.8 [12], designed for robust optimization problems in the MATLAB environment. In addition, the solver engine MOSEK version 6 was used in conjunction with ROME.

The solution proposed LAT and IAT to be set at 24.26 weeks and 14.6 weeks respectively, yielding a θ of 62.51 %. To test and compare the system performance under the original and proposed settings, 100 realizations of the uncertain parameters were randomly generated by assuming a uniform probability distribution with supports as in Table 2. Figure 2 and 3 show the behavior of the four state variables under the original settings and the proposed settings, respectively.

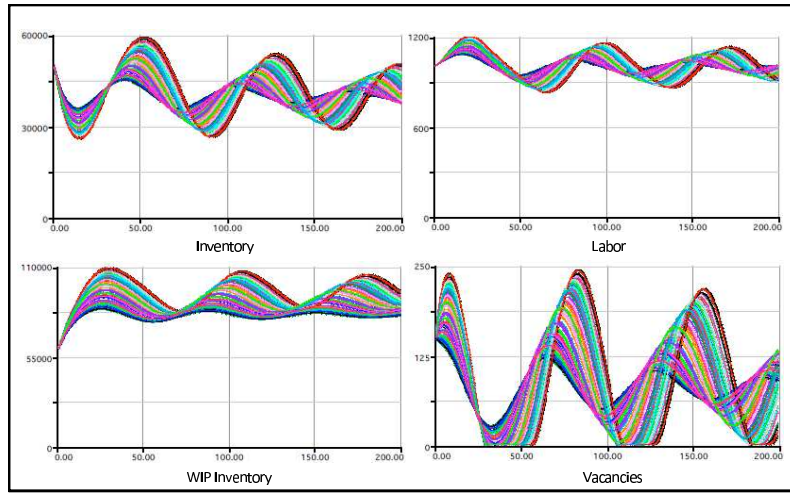


Fig. 2 Workforce inventory dynamics: original parameter settings

The behaviors of the state variables in Figure 2 coincided with what the dominant eigenvalues in §4.1 indicated. Indeed, the current parameter settings produced dampening oscillations for all state variables. The rates of which are likewise observed to be weak such that there have been no significant reductions in the fluctuations of the state levels throughout the simulation period.

Upon the implementation of the new settings for LAT and IAT , Figure 3 shows improvements in the stability of the system via stronger dampening of these fluctuations. The most significant improvement is observed from the behavior of vacancies. This indicates a reduction in the fluctuations of vacancy creation and closure rates, which led to fewer changes in the workforce levels of the system. These observations are attributed to the decrease in the variability of the WIP and finished goods inventory, wherein a more stable production (i.e. adjustments in the production rates/inventory levels) required fewer changes in the workforce levels of the system.

Table 3 shows the 95% intervals of the eigenvalues under the new settings evaluated using the simulated realizations. The upper bound for the real parts of λ_2 and λ_3 is -0.022 , compared to the initial value of -0.0095 . This shows that it has been made more

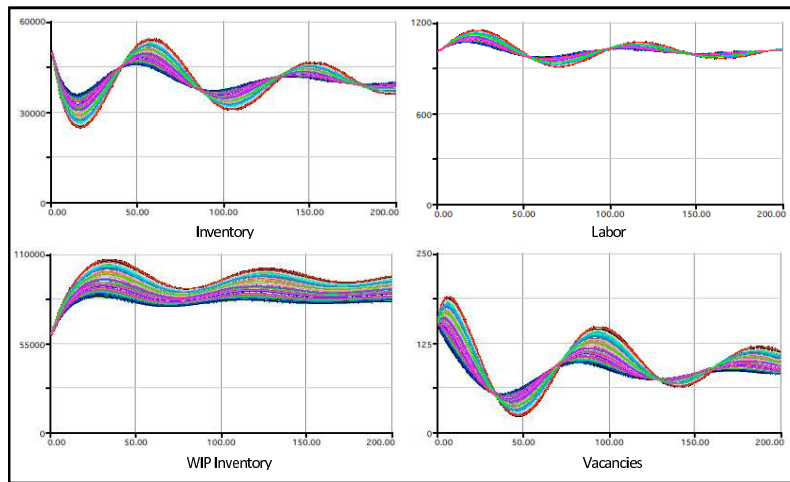


Fig. 3 Workforce inventory dynamics: proposed parameter settings

negative, thereby verifying the observed dampening of oscillations in the state variables. The table also shows that the imaginary part decreased from an initial value of -0.0988 to a range of 0.0905 to 0.0913, which indicates a decrease in the frequency of oscillations. All of which points to the achievement of the eigenvalue targets. Furthermore, these observations are all aligned with the generated behaviors of the state variables as shown in Figure 3. Therefore, the new parameter settings have not only achieved the desired behaviors for the workforce-inventory system but also made it robust to uncertainties.

$\text{Re}(\lambda_2, \lambda_3)$	$\text{Imag}(\lambda_2, \lambda_3)$	λ_1	λ_4
$[-0.02194, -0.02186]$	$[0.0905, 0.0913]$	$[-0.3729, -0.3517]$	$[-0.1102, -0.1088]$

Table 3 95% Intervals for Eigenvalues Under Proposed Parameter Design

5 Conclusion

This work proposed a satisficing approach to decision-making by maximizing a system's tolerance to uncertainty. The key advantages of the underlying mathematical programming model are its ease in computation and that it does not require specific probability distributions to be assumed for the uncertain variables. The model also allows the user to specify the set of targets to be satisfied. The computed solution is then one that seeks to achieve these targets as well as possible under uncertainty. The computational study demonstrated that the application of the proposed optimization model is able to improve the behavior of the system significantly under uncertainty.

Acknowledgements This work is supported by grant NUS R-266-000-058-112.

References

1. Ben-Tal, A., Nemirovski, A., Robust solutions of linear programming problems contaminated with uncertain data, *Mathematical Programming*, 88, 411-424 (2000)
2. Bertsimas D., Sim, M., The price of robustness. *Operations Research*, 52, 35-53 (2004)
3. Ben-Tal, A., Nemirovski, A., Robust solutions to uncertain programs. *Operations Research Letters*, 25, 1-13 (1999)
4. Bordley, R., LiCalzi, M., Decision analysis using target instead of utility functions, *Decisions in Economics and Finance*, 23, 53-74 (2000)
5. Brown, D., Sim, M., Satisficing measures for analysis of risky positions, *Management Science*, 55(1), 71-84 (2009)
6. Castagnoli, E., LiCalzi, M., Expected utility without utility, *Theory and Decisions*, 41, 281-301 (1996)
7. Charnes, A., Cooper, W.W., Deterministic Equivalents for Optimizing and Satisficing under Chance Constraints, *Operations Research*, 11(1), 18-39 (1963)
8. Chen, X., Sim, M., Goal driven optimization, *Operations Research*, 57, 342-357 (2009)
9. Diecidue, E., van de Ven, J., Aspiration level, probability of success and expected utility, *International Economic Review*, 49, 683-700 (2008)
10. Dong, Z., Pang, C., Zhang, P., Power system sensitivity analysis for probabilistic small signal stability assessment in a deregulated environment, *International Journal of Control, Automation, and Systems*, 3(2) (special edition), 355-362 (2005)
11. Forrester, N. A Dynamics Synthesis of Basic Macroeconomic Policy: Implications for Stabilizing Policy Analysis, PhD Thesis, Sloan School of Management, MIT: Cambridge, MA (1982)
12. Goh, J., Sim, M., Robust Optimization Made Easy with ROME, accepted for publication in *Operations Research* (2010)
13. Kampmann, C., Feedback loop gains and system behavior, In proceedings of the International System Dynamics Conference (pp. 260-263), System Dynamics Society: Cambridge, MA (1996)
14. Kampmann, C., Oliva, R., Loop eigenvalue elasticity analysis: three case studies, *System Dynamics Review*, 22(2), 146-162 (2006)
15. Luenberger, D., Introduction to Dynamic Systems: Theory, Models and Applications. John Wiley & Sons: Canada (1979)
16. Pagola, F., Perez-Arriaga, I., Verghese, G., On sensitivities, residues and participations: applications to oscillatory stability analysis and control, *IEEE Trans. on Power Systems*, 4(1), 278-285 (1989)
17. Payne, J.W., Laughunn, D.J., Crum, R., Translation of gambles and aspiration level effects in risky choice behavior, *Management Science*, 26, 1039-1060 (1980)
18. Saleh, M., Oliva, R., Davidsen, P., Kampmann, C., A comprehensive analytical approach for policy analysis of system dynamics models, *European Journal of Operational Research*, 203, 673-683 (2010)
19. Simon, H., Theories of decision-making in economics and behavioral science, *The American Economic Review*, 49(3), 253-283 (1959)
20. Sterman, J., Business Dynamics: Systems Thinking and Modeling for a Complex World. Irwin/McGraw-Hill: Boston, MA (2000)

A Proof for Proposition 1

First, formulation $\mathcal{RC}$ can be constructed by considering each of the $i = 1, \dots, N$ constraints in (3). In particular, for the i^{th} constraint, it suffices that:

$$A_i \cdot \mathbf{x} + \max_{\mathbf{z} \in F(\theta)} \{B_i \cdot \mathbf{z}\} \leq \tau \quad (12)$$

where A_i and B_i denote the i^{th} row of A and B respectively. The maximization in the left-hand side can be written explicitly as follows.

$$\max B_i \cdot \mathbf{z} \text{ s.t. } -\theta \bar{\mathbf{z}} \leq \mathbf{z} \leq \theta \bar{\mathbf{z}}, R\mathbf{z} \leq \mathbf{e} \quad (13)$$

The above is a linear optimization problem and the dual formulation can be written as:

$$\min(P_i + Q_i) \cdot \theta \bar{z} + S_i \cdot e \quad (14)$$

$$\text{s.t. } P_i - Q_i + S_i \cdot R \geq B_i, P_i, Q_i, S_i \geq 0 \quad (15)$$

where $P_i, Q_i \in \mathbb{R}^L$ and $S_i \in \mathbb{R}^W$ denote the variables in the dual formulation. Let $P = [P_1^\top, \dots, P_N^\top]$, $Q = [Q_1^\top, \dots, Q_N^\top]$, and $S = [S_1^\top, \dots, S_N^\top]$. The formulation $\mathcal{RC}$ is then derived by replacing, for all $i = 1, \dots, N$, the maximization term in (12) with the objective function value in (14) for each $i = 1, \dots, N$, thus obtaining (4), and augmenting (15) into $\mathcal{RTA}$, obtaining (5) and (6). Since both formulations are identical in the objective functions, it suffices to show that for any given θ , the feasible space of $\mathbf{x}$ in each of the formulations are the same.

1. $\mathcal{RTA} \subseteq \mathcal{RC}$: Suppose $\mathbf{x} \in \mathcal{RTA}$. Consider each $i = 1, \dots, N$. Since (13) is both feasible and bounded in the objective, its dual (14) and (15) is also feasible and bounded. Then by the strong duality of linear programming, there must exist P_i^* , Q_i^* and S_i^* from the dual space (15) so that $(P_i^* + Q_i^*) \cdot \theta \bar{z} + S_i^* \cdot e = \max \{B_i \cdot \mathbf{z}\} \leq \tau_i - A_i \cdot \mathbf{x}$ and hence $\mathbf{x} \in \mathcal{RC}$.
2. $\mathcal{RC} \subseteq \mathcal{RTA}$: Suppose $\mathbf{x} \in \mathcal{RC}$. By weak duality we have

$$\max_{\mathbf{z} \in F(\theta)} \{B_i \cdot \mathbf{z}\} \leq ((P_i + Q_i) \cdot \theta \bar{z} + S_i \cdot e \leq \tau_i - A_i \cdot \mathbf{x}),$$

for any P_i, Q_i and S_i from (15). Consequently $\mathbf{x} \in \mathcal{RTA}$, and this completes the equivalence of the two formulations. $\square$

B Proof for Proposition 2

Proof: We show that $\theta(\bar{v})$ defined in (7), where the random variable

$$\bar{v} = \min_{i=1, \dots, N} -\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z} + \tau_i,$$

satisfies each axiom.

1. *Attainment content.*

$$\begin{aligned} \text{Given :} \quad & \bar{v} \geq 0 \\ \Rightarrow \quad & \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z}\} + \tau_i \geq 0 \quad \forall \mathbf{z} \in F(\theta = 1) \\ \Rightarrow \quad & \min_{\mathbf{z} \in F(\theta)} \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z}\} + \tau_i \geq 0 \quad \forall \theta \in [0, 1] \\ \Rightarrow \quad & \theta(\bar{v}) = 1 \end{aligned}$$

where the last inequality follows from noting that $F(\theta') \subseteq F(\theta)$ whenever $\theta' \leq \theta$.

2. *Non-attainment apathy.*

$$\begin{aligned} \text{Given :} \quad & \bar{v} < 0 \\ \Rightarrow \quad & \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z}\} + \tau_i < 0 \quad \forall \mathbf{z} \in F(\theta), \forall \theta \in [0, 1] \\ \Rightarrow \quad & \theta(\bar{v}) = 0 \quad (\text{by definition in (7)}). \end{aligned}$$

3. *Monotonicity.* Given $\mathbf{x}$ and $\mathbf{x}'$, and where

$$\bar{v}' = \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x}' - \mathbf{B}_i \mathbf{z}\} + \tau_i,$$

we have that:

$$\begin{aligned} & \bar{v} \geq \bar{v}' \\ \Rightarrow \quad & \forall \mathbf{z} \in F(\theta = 1), \\ & \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z}\} + \tau_i \geq \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x}' - \mathbf{B}_i \mathbf{z}\} + \tau_i \\ \Rightarrow \quad & \forall \mathbf{z} \in F(\theta = \theta(\bar{v}')), \\ & \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z}\} + \tau_i \geq \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x}' - \mathbf{B}_i \mathbf{z}\} + \tau_i \geq 0 \\ \Rightarrow \quad & \theta(\bar{v}) \geq \theta(\bar{v}') \end{aligned}$$

4. *Right continuity.* We proceed by showing that the below must hold simultaneously:

$$\lim_{\alpha \downarrow 0} \theta(\tilde{v} + \alpha) \leq \theta(\tilde{v}) \quad (16)$$

$$\lim_{\alpha \downarrow 0} \theta(\tilde{v} + \alpha) \geq \theta(\tilde{v}) \quad (17)$$

We show (16) by contradiction. Suppose $\lim_{\alpha \downarrow 0} \theta(\tilde{v} + \alpha) = \theta(\tilde{v}) + \epsilon$ for some $\epsilon > 0$. Then we have that:

$$\min_{\mathbf{z} \in F(\theta(\tilde{v}) + \epsilon)} \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z}\} + \tau_i + \alpha \geq 0 \quad \forall \alpha > 0 \quad (18)$$

Furthermore, by definition of $\theta(\tilde{v})$ we must have:

$$\min_{\mathbf{z} \in F(\theta(\tilde{v}) + \epsilon)} \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z}\} + \tau_i < 0$$

Clearly, by choosing any $\Delta > 0$ and such that

$$\Delta < - \min_{\mathbf{z} \in F(\theta(\tilde{v}) + \epsilon)} \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z}\} + \tau_i,$$

we can have

$$\alpha = - \min_{\mathbf{z} \in F(\theta(\tilde{v}) + \epsilon)} \min_{i=1, \dots, N} \{-\mathbf{A}_i \mathbf{x} - \mathbf{B}_i \mathbf{z}\} + \tau_i - \Delta > 0$$

which provides the required contradiction in (18).

Finally, (17) is straightforward to show by applying monotonicity of θ . That is, we must have that $\theta(\tilde{v} + \alpha) \geq \theta(\tilde{v})$ for all $\alpha > 0$. $\square$

C Supplementary Material for the Workforce-Inventory System

C.1 State-space model

Consider the following dynamic system defined through the set of ordinary differential equations:

$$\begin{aligned} \dot{\mathbf{y}} &= F(\mathbf{y}, \mathbf{w}, \mathbf{u}), \\ 0 &= G(\mathbf{y}, \mathbf{w}, \mathbf{u}), \end{aligned}$$

where $\mathbf{y}$ is a vector of the state variables, $\mathbf{w}$ is the vector of auxiliary variables, $\mathbf{u}$ are the exogenous variables, and F and G define functions of these variables.

For performing eigenvalue analysis, the following linear time-invariant state-space models are widely accepted [16] and can be obtained via first-order Taylor approximation of the differential equations.

$$\begin{bmatrix} \dot{\mathbf{y}} \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} D_{yy} & D_{yw} \\ D_{wy} & D_{ww} \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{w} \end{bmatrix}$$

In the state-space equations, D_{yy} , D_{yw} , D_{wy} and D_{ww} are partial derivatives of the differential equations with respect to the state variables and auxiliary variables in the system (i.e. $D_{yy} = \partial \dot{\mathbf{y}} / \partial \mathbf{y}$ and $D_{yw} = \partial \dot{\mathbf{y}} / \partial \mathbf{w}$). The state matrix D is then computed as follows:

$$\begin{aligned} D &= D_{yy} + D_{yw} [I - D_{ww}]^{-1} D_{wy} \\ &= \frac{\partial \dot{\mathbf{y}}}{\partial \mathbf{y}} + \frac{\partial \dot{\mathbf{y}}}{\partial \mathbf{w}} \left[I - \frac{\partial \mathbf{w}}{\partial \mathbf{w}} \right]^{-1} \frac{\partial \mathbf{w}}{\partial \mathbf{y}} \end{aligned}$$

The characteristic polynomial of the state matrix D is defined by $p(\lambda) = \det[D - \lambda I]$, where λ denotes the variables and I is an identity matrix. $p(\lambda)$ is a polynomial of degree n in λ , the roots of which are defined as the eigenvalues of the state matrix ([15]).

C.2 State level constraints

We can specify state level requirements using a discretized time approximation of the dynamic system. For a given $\mathbf{x}$ and $\mathbf{z}$, the changes in system output level $\mathbf{y}(t)$ at time t , where $t = 1, \dots, T$ about some nominal output level $\bar{\mathbf{y}}(t)$ due to parameter variations in D can be approximated as:

$$\begin{aligned} y(t) &\approx y(t-1) + \Delta D \bar{y}(t-1) + Dy(t-1) \\ &\approx y(t-1) + \frac{\partial D}{\partial \mathbf{x}} \bar{y}(t-1) \mathbf{x} + \frac{\partial D}{\partial \mathbf{z}} \bar{y}(t-1) \mathbf{z} + \frac{\partial D}{\partial \mathbf{xz}} \bar{y}(t-1) \mathbf{xz} + Dy(t-1) \\ &= (I + D)y(t-1) + \left(\frac{\partial D}{\partial \mathbf{x}} \mathbf{x} + \frac{\partial D}{\partial \mathbf{z}} \mathbf{z} + \frac{\partial D}{\partial \mathbf{xz}} \mathbf{xz} \right) \bar{y}(t-1) \end{aligned}$$

where $\frac{\partial D}{\partial \mathbf{x}}$, $\frac{\partial D}{\partial \mathbf{z}}$ and $\frac{\partial D}{\partial \mathbf{xz}}$ refer to the partial derivatives of the elements of D with respect to $\mathbf{x}$, $\mathbf{z}$ and $\mathbf{xz}$ evaluated at the nominal operating points. Applying the above repeatedly for $\mathbf{y}(t-1), \dots, \mathbf{y}(1)$, and defining $\mathbf{y}(0) = 0$, this can be rewritten as,

$$y(t) = \left(\frac{\partial D}{\partial \mathbf{x}} \mathbf{x} + \frac{\partial D}{\partial \mathbf{z}} \mathbf{z} + \frac{\partial D}{\partial \mathbf{xz}} \mathbf{xz} \right) \left[\sum_{t'=0}^{t-1} (I + D)^{t-1-t'} y(t') \right] \quad \forall t = 1, \dots, T$$

Clearly this can be written in the form of (3) by defining

$$A = \left[\frac{\partial D}{\partial x_1} \cdot \left[\sum_{t'=0}^{t-1} (I + D)^{t-1-t'} \bar{\mathbf{y}}(t') \right], \dots, \frac{\partial D}{\partial x_M} \cdot \left[\sum_{t'=0}^{t-1} (I + D)^{t-1-t'} \bar{\mathbf{y}}(t') \right] \right]$$

and similarly

$$B = \left[\frac{\partial D}{\partial z_1} \cdot \left[\sum_{t'=0}^{t-1} (I + D)^{t-1-t'} \bar{\mathbf{y}}(t') \right], \dots, \frac{\partial D}{\partial z_L} \cdot \left[\sum_{t'=0}^{t-1} (I + D)^{t-1-t'} \bar{\mathbf{y}}(t') \right] \right]$$

Multi-mode Resource-Constrained Project Scheduling of Semiconductor Equipment Install/Qual Process

Cheng, Junzilan · Fowler, John W · Hertzler,
Erik F · Huff, Jerry D · Kempf, Karl G

Abstract Ramping up a semiconductor wafer fabrication (fab) facility is a challenging endeavor. One of the key components of this process is scheduling multiple resources to install and qualify capital intensive and sophisticated manufacturing equipment which is necessary to produce integrated circuits. We approach this problem by modeling it as a resource-constrained project scheduling problem (RCPSP). We extend the classical RCPSP model to handle special aspects of the semiconductor environment such as multiple resource calendars, time-varying resource constraints, alternative activity processing modes, controllable activity durations, etc. In order to illustrate the problem solving approach, we provide a mixed integer linear programming model and a simple computational example. We consider two objectives in the research: reducing project makespan and minimizing resource costs.

1 Introduction

Factory ramping is the process of installing and qualifying (install/qual) manufacturing equipment which must be purchased from equipment suppliers before using them in the wafer fabrication process. The velocity of ramping up a wafer fab can impact a semiconductor company's bottom line by determining how fast the next generation product

Cheng, Junzilan(Felix)
Arizona State University
E-mail: Felix.Cheng@asu.edu

Fowler, John W
Arizona State University
E-mail: John.Fowler@asu.edu

Hertzler, Erik F
Intel Corporation
E-mail: Erik.F.Hertzler@Intel.com

Huff, Jerry D
Arizona State University
E-mail: Jerry.huff@asu.edu

Kempf, Karl G
Intel Corporation
E-mail: karl.g.kempf@intel.com

can be made available to customers. Further, a better coordinated install/qual schedule can shorten the capital equipment supply chain and delay decision-making which can diminish the variance in planned factory capacity based on predicted customer demand.

Equipment install/qual is a set of serial processes including: *Physical Installation* (PI), *Supplier Qualification* (SQ) and *Company Qualification* (CQ) as illustrated in Figure 1.

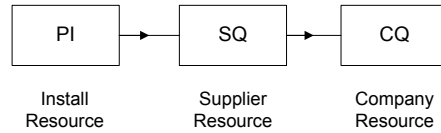


Fig. 1: The Install/qual process

There are several challenges that prevent schedulers from improving the install/qual schedule. First, the three distinct install/qual processes each require a different set of resources. PI is the process of installing the infrastructure needed by the equipment(e.g. pipes for water) and then installing the actual equipment. Equipment installation is conducted by Install Resources who are contracted from outside the company; SQ is the qualifying process carried out by equipment suppliers: Supplier Resources. Since equipment may be supplied by different suppliers, there are multiple supplier resources; Last, the company has its own qualifying engineers (Company Resource) responsible for CQ process.

As illustrated in Table 1, different resources have different working calendars. Multiple different working calendars may cause schedule overlap or time gaps which is also a challenge in the install/qual scheduling problem.

Table 1: Inconsistent Resource Calendars

Resource	Working Calendar
Install Resource	8 hours/day $\times$ 5 days/week
Supplier Resources	10 hours/day $\times$ 4 days/week
Company Resource	24 hours/day $\times$ 7 days/week

Further, there exist alternative modes for executing an activity such as hiring a skillful but expensive team A vs. a less experienced but lower priced team B. Also, some activity durations are controllable through the amount of resources allocated to these activities. For example, 2 engineers finish a given task in 4 days while 3 equally experienced engineers might reduce that duration to 3 days.

The objective of this research is to find better scheduling methods to handle these practical challenges. In the next section, we review three related streams of academic research. In sections 3 and 4, the mathematical formulation and a small computational example are discussed. Conclusions and future research are addressed in section 5.

2 Literature Review

Three related academic research streams are reviewed in this section: the resource constrained project scheduling problem (RCPSP), scheduling problems with controllable processing times, and scheduling problems with flexible resources.

2.1 Resource-constraint project scheduling problem

Research related to the resource-constrained project scheduling problem (RCPSP) dates back to the 1960's ([11], [12]). The RCPSP can be stated as follows: A given project contains a set of activities (tasks) which are connected by direct precedence relations among activities. Processing an activity requires a certain amount of one or multiple types of resources during its process duration. Each type of resource has capacity limits in individual schedule time slots or over the project schedule horizon. The RCPSP objective is to find a feasible solution (schedule does not violate precedence constraints or resource capacity constraints) to achieve the minimum project makespan (project duration).

Resources can be classified into different categories according to different criteria. The most commonly used classification criteria are based on constraints on resources. *Renewable resources* are limited per time-unit (e.g. manpower, machines and space); *Nonrenewable resources* are limited over the entire project duration (e.g. budget/money, energy and raw material).

In order to handle special applications, several RCPSP extensions have been developed. [17] provides a good summary of RCPSP extensions. Major RCPSP extensions that relate to our research include: multi-mode RCPSP which considers cases when there are multiple modes in processing some of the activities in a project; RCPSP with time-varying resource constraints deals with situations when resource limits change over time; Bi-criteria(multi-criteria) RCPSP is RCPSP with two(multiple) objectives; Preemptive RCPSP allows activities to be preempted during processing. Another review of RCPSP research can also be found at [2].

2.2 Scheduling problems with controllable processing times

Research related to scheduling problems with controllable processing times has been considered for several machine scheduling environments. In these research efforts, the constraint of constant job processing times from classical deterministic machine scheduling is replaced by controllable processing times through resource allocations. Thus, resource allocation strategy becomes a decision that must be made in addition to determining job sequence. A review of this literature can be found at [15]. Both linear and convex functions are discussed in most previous research regarding job processing times and resource amounts. Similar ideas can be used in modeling some install/qual activities where activity durations are controllable through resources. However, much of the previous research is for the single machine environment and continuous resources. Only special cases of flow shops have been studied: [9] and [14] which focus on two machine flow shops; [4], [10] and [13] which consider permutation flow shop problems. The install/qual scheduling problem can be treated as a flexible flow shop with discrete types of resources.

2.3 Scheduling problems with flexible resources

The concept of flexible resources deals with scheduling problems where resources are cross-trained and can be treated as different types. The primary focus of this research is to optimize the resource allocation strategy which is similar to the install/qual problem. Previous research of this problem can be found in [5], [6] and [7] .

3 Methodology and Modeling Approach

In this section, we first give the problem definition and then discuss the mathematical formulation of the problem.

3.1 Problem Definition

Mathematically, a resource-constrained project scheduling problem can be stated as follows (notation used can be found in Table 2). In a project network in **AoN** (activity on node) format $G(V, E)$, there are a set of nodes V corresponding to activities in the project and a set of directed arcs E that represent precedence relations among activities. Activities in V are numbered from 1 to n ($|V| = n$). Even though how activities are numbered does not matter as long as there is a one-to-one mapping between node numbers in the network and activities in the project, it will make the project network $G(V, E)$ easier to interpret if predecessor nodes are numbered smaller than successor nodes. For network completeness, there exists a dummy start node 0 and a dummy finish node $n + 1$, both with no resource requirements and zero duration. In a manufacturing equipment install/qual project, install/qual operations of equipment are the activities in the project.

There are two types of precedence constraints in this problem. First, there are the constraints that represent the sequences of install/qual operations (*Physical Installation*, *Supplier Qualification* and *Company Qualification*) for an individual piece of equipment. Second, there can be precedence constraints between different individual pieces of equipment. These precedence constraints are represented by directed arcs in the project network.

Install resources, supplier resources and company resources belong to renewable resources $\mathbb{R}$. The only non-renewable resource $\mathbb{R}^n$ is the total resource cost budget B . For each resource type $k \in \mathbb{R}$, we define a "resource profile" as a function that specifies the upper bound (U_{kt}) and lower bound (L_{kt}) on resource availability at each time slot. Using the idea of different "resource profiles", we can compare schedules with different working calendar options such as: 8 hours/day $\times$ 5 days/week vs. 24 hours/day $\times$ 7 days/week. We can also change the upper and lower bound on resource constraints to answer questions like whether the company should hire more resources given the extra resource cost. Any change of "resource profile" will potentially affect every activity in the project.

Contrary to "resource profile", the ideas of "multi-mode" and "controllable processing times" are applied only on individual activities. For a given activity in the project, "multi-mode" and "controllable processing time" can be used to model the case of multiple resource teams or how many people should be allocated to a task. Each activity $j \in V$ has a processing duration p_j that requires a certain amount of

required renewable and/or non-renewable resources. In the multi-mode case, p_j^m is the processing duration for activity j under mode m .

Since resources might not be available during some time periods (such as weekends), activities are allowed to be interrupted and split when resources become unavailable. With activity splitting, started activities are allowed to pause when resources are temporarily unavailable and resume at later time periods.

Table 2: Mathematical Notation

Paremters	Description
j	Activity(task) j in the project
t	Time t
k	Resource type
m	Activity(task) processing mode
M_j	The set of available processing modes for activity(task) j
c_k	Unit cost for resource type k
p_j^m	Processing duration for activity j under mode m
r_{jk}^m	Required amount for resource type k for activity j under mode m
U_{kt}	Upper bound on the available amount of resource type k at time t
L_{kt}	Lower bound on the available amount of resource type k at time t
S_j	The start time of activity j , the beginning of that time period
F_j	The finish time of activity j , the beginning of that time period
V	The set of activities(tasks) in the project $1, 2, \dots, n$
$\mathbb{R}^r$	The set of renewable resources
$\mathbb{R}^n$	The set of non-renewable resources
C_{max}	The makespan of the project, the completion time of the last activity
B	Budget for total resource costs
T	The maximum project scheduling time
TC	The total resource cost
$G(V, E)$	Network G consisting of nodes V and arcs E
$arc(i, j)$	Directed arc connecting node i to node j
$pred(j)$	The set of predecessor activities of activity j , $pred(0) = \emptyset$
$succ(j)$	The set of successor activities of activity j , $succ(n+1) = \emptyset$

3.2 Mathematical Formulation

The mixed integer programming model we propose is adopted from [3]. First, the decision variables are listed as follows:

$$y_j^m = \begin{cases} 1 & \text{if activity } j \text{ is being processed in mode } m \in M_j, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

in which M_j is the set of all possible processing modes for activity j .

$$x_{jt}^m = \begin{cases} 1 & \text{if activity } j \text{ is being processed in mode } m \in M_j \text{ at time } t, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

There are two sets of supporting decision variables: S_j and F_j represent the starting time and finish time of activity j respectively, $j \in V$. To avoid unnecessary confusion, we mark both S_j and F_j at the beginning of each time period.

The first objective is to minimize the install/qual project makespan:

$$\min \quad C_{max} \quad (3)$$

which is equivalent to minimizing the finish time of the last activity (dummy finish activity) $n + 1$:

$$\min F_{n+1} \quad (4)$$

The second objective is to minimize the total resource cost TC which can be calculated from the total resource usage and the unit resource cost:

$$\min TC = \sum_{t=1}^T \sum_{k \in \mathbb{R}^r} c_k \cdot \sum_{j \in V} \sum_{m \in M_j} r_{jk}^m \cdot x_j t^m \quad (5)$$

In this research, we use the ϵ -constraint method [16] to convert the bi-criteria optimization case into a single objective optimization problem by setting a total budget limit on the resource cost in constraints as follows:

$$TC \leq B \quad (6)$$

For every activity $j \in V$, only one mode can be selected:

$$\sum_{m \in M_j} y_j^m = 1, \quad \forall j \in V \quad (7)$$

For each activity j , if mode m is selected, the total time for activity j is being executed equals to the activity duration which is p_j^m :

$$\sum_{t=1}^T x_{jt}^m = p_j^m \cdot y_j^m, \quad \forall j \in V, m \in M_j \quad (8)$$

And this set of constraints can also satisfy that if $x_{jt}^m = 1$ for some time t , then y_j^m has to be 1; and if $y_j^m = 0$ which means mode m is not selected for activity j , then $x_{jt}^m = 0$ for all time periods t .

The precedence constraints for all arcs in the precedence network should be satisfied:

$$S_j \leq x_{jt}^m \cdot t + M(1 - x_{jt}^m), \quad m \in M_j, \forall j \in V, t = 1, 2, \dots, T \quad (9)$$

$$F_j \geq x_{jt}^m \cdot t, \quad m \in M_j, \forall j \in V, t = 1, 2, \dots, T \quad (10)$$

in which M is a large enough value. For example, we can set M to be the summation of all activity durations among all modes for all activities. Since we mark activity start time and finish time at the beginning of each scheduling time slot, we have the following set of constraints:

$$F_i \leq S_j - 1, \quad \forall \text{arc}(i, j) \text{ in } G, m \in M_j, j \in V, t = 1, 2, \dots, T \quad (11)$$

Resource capacity restrictions have to be satisfied at each integer time point and for each renewable resource:

$$L_{kt} \leq \sum_{j \in V} \sum_{m \in M_j} r_{jk}^m \cdot x_{jt}^m \leq U_{kt}, \quad \forall k \in \mathbb{R}^r, t = 1, 2, \dots, T \quad (12)$$

The following are the integrality and non-negativity constraints:

$$y_j^m \in \{0, 1\} \quad (13)$$

$$x_{jt}^m \in \{0, 1\} \quad (14)$$

$$S_j \geq 0 \quad (15)$$

$$F_j \geq 0 \quad (16)$$

Since the total resource cost TC is limited by resource budget B whose value is determined by the scheduler, the model can find different shortest makespan schedule(s) under different given resource budget limits. Thus, the overall install/qual scheduling problem objective is to find schedules that are Pareto Optimal regarding both project makespan and resource costs. Definition of Pareto Optimal solutions and bicriteria optimization can be found in [8] and [16]. An install/qual schedule can be specified as the starting time of each activity $(S_1, S_2, \dots, S_n)$, the finishing time of each activity $(F_1, F_2, \dots, F_n)$, the processing mode for each activity $(m_1, m_2, \dots, m_n)$ and whether an activity j is being processed at time slot t with mode m .

4 Computational Experiment

In this section, a simple install/qual scheduling problem instance is studied. As illustrated in the following project network Figure 2, five nodes (1, 2, 3, 4, 5) represent five pieces of manufacturing equipment (1, 2, 3, 4, 5) that need to be installed and qualified respectively. Nodes 0 and 6 are the dummy start and dummy finish tasks, both have zero task durations and no resource requirements. Precedence relationships between the pieces of equipment are indicated by the directed arcs between nodes.

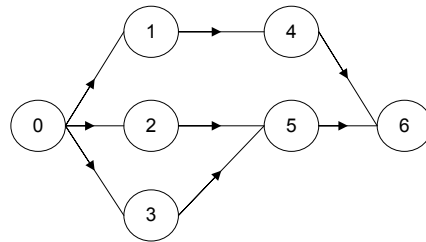


Fig. 2: An Install/qual project

To install/qual each piece of equipment, three serial tasks (PI, SQ, CQ) should be finished and each task requires different types of resources. Thus, the overall project network is illustrated in Figure 3.

There are four types of resources that are involved in this install/qual scheduling problem example. Resource type 1 and type 4 are shared by all PI and CQ activities respectively; regardless the equipment number. However, different equipment suppliers are only in charge of their own equipment. In this example, equipment 1 and 4 are from supplier 1 and represented by resource type 2; equipment 2, 3 and 5 are from supplier 2 and represented by resource type 3. All possible activity processing modes are provided in Table 3. For a given processing mode, the activity number, resource

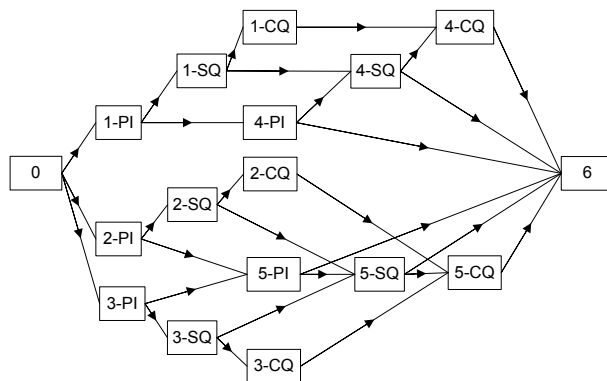


Fig. 3: The real project network

type(s), resource amount, duration and cost for choosing that mode are specified. Even though the model can actually handle multiple types of resources associated with one activity, we only consider one type of resource corresponding to one activity in this example.

Resource profiles for these four types of resources are provided in Figure 4. The overall scheduling time horizon is 90 days which is carefully chosen to make sure it is longer than the maximum possible scheduling time. It is worthwhile to mention that resource type 1 does not work during weekends; resources 2 and 3 do not work on Fridays, Saturdays and Sundays; and resource 4 works all seven days per week. Resource amounts vary over time as well.

Solutions are provided in Table 4 and plotted in Figure 5 regarding two objectives: project makespan and total resource cost. A clear trade-off relation can be observed in Figure 5 which can help decision-making for schedulers in balancing install/qual duration with resource investments.

5 Conclusion and Future Research

In this paper, we consider a project scheduling problem for the equipment installation and qualification process in the semiconductor manufacturing environment. We briefly reviewed three related academic scheduling research streams: Resource-constrained project scheduling (RCPSP), scheduling problems with controllable processing times and scheduling problems with flexible resources. The methodology we propose is to combine the concepts of controllable processing times and flexible resources with the classical RCPSP. We extend RCPSP to handle special aspects in the install/qual scheduling problem to include: multiple resources with multiple modes and different working calendars, time-varying resource constraints and two scheduling objectives. Our methodology was illustrated by a small problem instance.

Regarding the two objectives, install/qual project duration and total resource cost, several Pareto Optimal solutions and the efficient frontier were generated. However, even for a problem instance whose size is as small as 5 pieces of equipment and 3 or fewer modes for each equipment, the computational time for generating some solutions took more than 7 hours. This is clearly not efficient enough. Since classical RCPSP

Table 3: Available Activity Processing Modes

	Act	RType	Mo	Dur	RAmount	Cost
0	0	0	0	0	0	0
1-PI	1	1	1	2	3	1
1-PI	1	1	2	1	2	3
1-PI	1	1	3	3	1	2
2-PI	2	1	1	2	3	2
2-PI	2	1	2	1	2	3
2-PI	2	1	3	4	2	1
3-PI	3	1	1	1	4	4
3-PI	3	1	2	2	3	3
3-PI	3	1	3	4	2	1
4-PI	4	1	1	1	3	3
4-PI	4	1	2	3	2	3
4-PI	4	1	3	4	1	2
5-PI	5	1	1	1	2	4
5-PI	5	1	2	2	2	2
5-PI	5	1	3	4	3	1
1-SQ	6	2	1	1	2	4
1-SQ	6	2	2	4	3	1
2-SQ	7	3	1	2	1	3
2-SQ	7	3	2	3	2	2
3-SQ	8	3	1	2	3	4
3-SQ	8	3	2	4	2	1
4-SQ	9	2	1	1	3	4
4-SQ	9	2	2	3	2	2
5-SQ	10	3	1	1	3	3
5-SQ	10	3	2	4	2	1
1-CQ	11	4	1	1	2	4
1-CQ	11	4	2	4	2	2
2-CQ	12	4	1	1	2	4
2-CQ	12	4	2	3	2	1
3-CQ	13	4	1	1	3	4
3-CQ	13	4	2	4	3	1
4-CQ	14	4	1	2	3	4
4-CQ	14	4	2	3	3	2
5-CQ	15	4	1	2	2	4
5-CQ	15	4	2	4	2	1
15	16	0	0	0	0	0

Table 4: Computational Results

Budget	20	21	22-23	24-26	28-29	30-31	32-33	35	36	37-39	40	42-55
Cmax	28	24	23	22	20	19	18	17	16	15	14	13

has been proven to be NP-hard in [1], the natural next step for this research is to develop heuristics and meta-heuristics to find near optimal solutions within a reasonable amount of computation time. We plan to use random-key based Genetic Algorithm in our ongoing research.

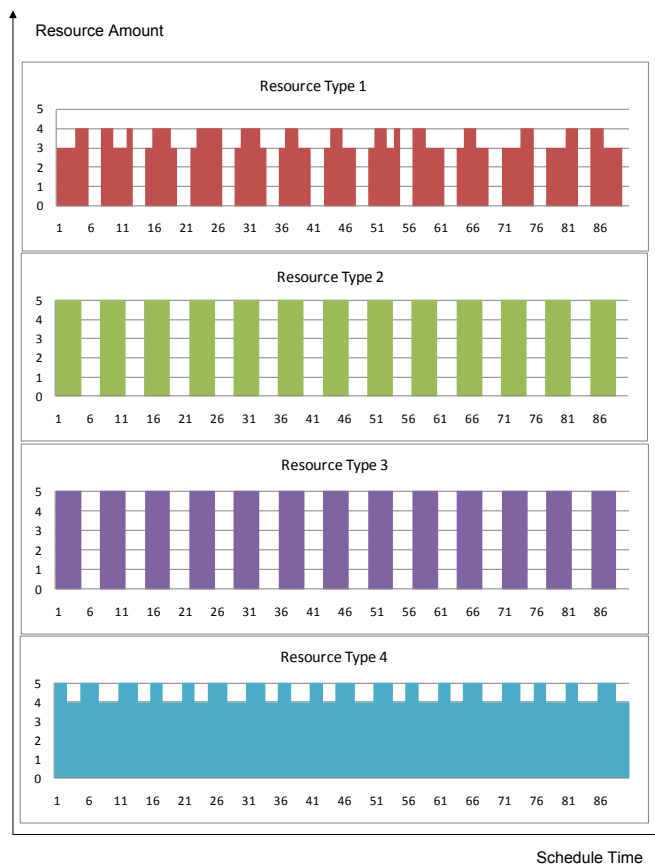


Fig. 4: The resource profiles

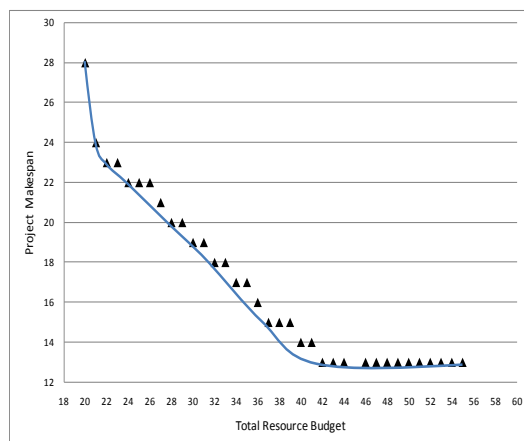


Fig. 5: The Sample Problem Solutions and the Efficient Frontier

References

1. J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.
2. P. Brucker, A. Drexel, R. Möhring, K. Neumann, and E. Pesch. A resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999. Compilation and indexing terms, Copyright 2010 Elsevier Inc.; M1: Compendex; undefined.
3. J. Buddhakulsomsiri and D. S. Kim. Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. *European Journal of Operational Research*, 178(2):374–390, 2007.
4. T. C. E. Cheng and A. Janiak. A permutation flow-shop scheduling problem with convex models of operation processing times. *Annals of Operations Research*, 96(1):39–60, 2000.
5. R. L. Daniels and J. B. Mazzola. Flow shop scheduling with resource flexibility. *Operations Research*, 42(3):504–522, 1994.
6. R. L. Daniels, J. B. Mazzola, and D. Shi. Flow shop scheduling with partial resource flexibility. *Management Science*, 50(5):658–669, 2004.
7. S. Dauzère-Pérès, W. Roux, and J. B. Lasserre. Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research*, 107(2):289–305, 1998.
8. M. Ehrgott. *Multicriteria optimization*. Springer Verlag, 2005.
9. A. Janiak. Minimization of resource consumption under a given deadline in the two-processor flow-shop scheduling problem. *Information processing letters*, 32(3):101–112, 1989.
10. A. Janiak and M. C. Portmann. Genetic algorithm for the permutation flow-shop scheduling problem with linear models of operations. *Annals of Operations Research*, 83:95–114, 1998.
11. T. J. R. Johnson. An algorithm for the resource constrained project scheduling problem, 1967.
12. T. J. R. Johnson. A branch and bound algorithm for the resource constrained project scheduling problem. Research report, 1967.
13. E. Nowicki. An approximation algorithm for the m-machine permutation flow shop scheduling problem with controllable processing times. *European Journal of Operational Research*, 70(3):342–349, 1993.
14. E. Nowicki and S. Zdrzaka. A two-machine flow shop scheduling problem with controllable job processing times. *European Journal of Operational Research*, 34(2):208–220, 1988.
15. D. Shabtay and G. Steiner. A survey of scheduling with controllable processing times. *Discrete Applied Mathematics*, 155(13):1643–1666, 2007.
16. V. Tkindt and J. C. Billaut. *Multicriteria scheduling: theory, models and algorithms*. Springer Verlag, 2006.
17. J. Węglarz, J. Józefowska, M. Mika, and G. Waligóra. Project scheduling with finite or infinite number of activity processing modes - a survey. *European Journal of Operational Research*, 208(3):177–205, 2011.

The Airport Baggage Sorting Station Allocation Problem

Amadeo Ascó · Jason A. D. Atkin ·
Edmund K. Burke

Abstract The appropriate assignment of airport resources can greatly affect the quality of service which airlines and airports provide to their customers. Good assignments can help airlines and airports to keep to published schedules, by minimising changes or delays waiting for resources to be free.

In this paper, we consider a resource allocation problem, namely the problem of assigning available baggage sorting stations to flights which have already been scheduled. A description and model for the problem are presented, illustrating the different objectives which have to be considered. A number of constructive algorithms for sorting station allocation are then presented and their effects are compared and contrasted for different numbers of sorting station that are available. In particular, it can be observed that the appropriate algorithm selection is highly dependent upon whether reductions in service times are permitted or not and upon the flight density in relation to the number of sorting stations.

Keywords: Airport Baggage Sorting Stations, Scheduling, Heuristics, Constructive Algorithms, Greedy Algorithm

1 Introduction

The mishandling of baggage in airports has been one of the more important issues for passengers for several years in both Europe and the U.S.A. It was ranked third in complaints after cancellations and delays in the 2009 report of the Air Transport Users Council (2009) and its importance was emphasized further in the April 2010 report of the Office of Aviation Enforcement and Proceedings (U.S. Department of Transportation (2010)) where over a hundred thousand baggage reports were logged,

Amadeo Ascó
School of Computer Science
University of Nottingham
E-mail: a.asco@cs.nott.ac.uk

Dr. Jason A. D. Atkin
School of Computer Science
University of Nottingham
E-mail: jaa@cs.nott.ac.uk

Prof. Edmund K. Burke
School of Computer Science
University of Nottingham
E-mail: ekb@cs.nott.ac.uk

ranking baggage complaints in second place. Expected increases in civil air traffic (ICAO (2010) and Federal Aviation Administration (2010)) will continue to increase the complexity of these problems.

Baggage which is checked in will travel through the baggage system to baggage sorting stations. Baggage handlers at the sorting stations will sort and load the baggage onto baggage carts or into special containers that go straight into the aircraft. These sorting stations are not usually co-located with the aircraft, so the pairing of sorting station to gate is important. Furthermore, baggage often arrives or accumulates at the sorting station before the aircraft arrive at the gate, so the sorting station is needed earlier than the gate, and usually for longer than the gate.

We present a model for the baggage sorting station allocation problem at airports, along with a consideration of the various objectives. The problem can be observed to be a multi-objective resource constrained allocation problem, where the aim is to allocate the limited baggage handling resources amongst the various flights which have to be serviced. Research into a similar problem was performed in Abdelghany et al (2006), but various questions were left unanswered. This paper aims to answer these questions and to perform a rigorous analysis of the effects and benefits of various different constructive algorithms for the problem, with a view to potentially utilising these to provide initial solutions for further search methods. The intention is not to determine the 'perfect' algorithm for constructing a sorting station allocation, but instead to understand the effects and trade-offs of different choices.

In the model considered here, which was presented by Abdelghany et al (2006), flights are first allocated to stands (i.e. the Airport Gate Assignment Problem (GAP) is solved) before the baggage allocation problem is considered. An example allocation of flights is shown in Figure 1, which can be considered to be a type of GANTT chart, where the vertical axis represents the stands and the horizontal axis shows the time of day. Each rectangle on the diagram represents a specific flight and shows the times that the flight will use the stand. Each stand is numbered: the first digit is the terminal number, the second digit is the pier number, and the last two digits are the individual stand identification. For example, the top row shows four flights assigned to stand 1101, which refers to terminal 1, pier 1 and stand 1.

The root of the problem for baggage sorting station allocation is that baggage sorting stations are required for longer than flights, so there can be no one-to-one correspondence between baggage sorting stations and stands, and ideal locations cannot be guaranteed. Indeed, ideally there should also be a buffer time between sorting station usages, to reduce the risk of small perturbations affecting the allocations and of the mixing of baggage between flights, but the contention for baggage sorting stations means that this sometimes has to be reduced or eliminated. One of the purposes of this paper is to better understand the way in which the potential reductions in buffer times affect the various algorithms.

There are a number of objectives to consider in the baggage sorting station allocation problem (for example, maximising the allocations, maximising available buffer times and allocating flights to the closest sorting stations) and these are in obvious conflict with each other. Any solution method needs to take this into account. In particular, different constructive algorithms will be observed in this paper to perform better for differing objectives. Hybridisation of the algorithms themselves or the appropriate utilisation or recombination of solutions from different algorithms may potentially lead to allocations which better reflect the overall objectives.

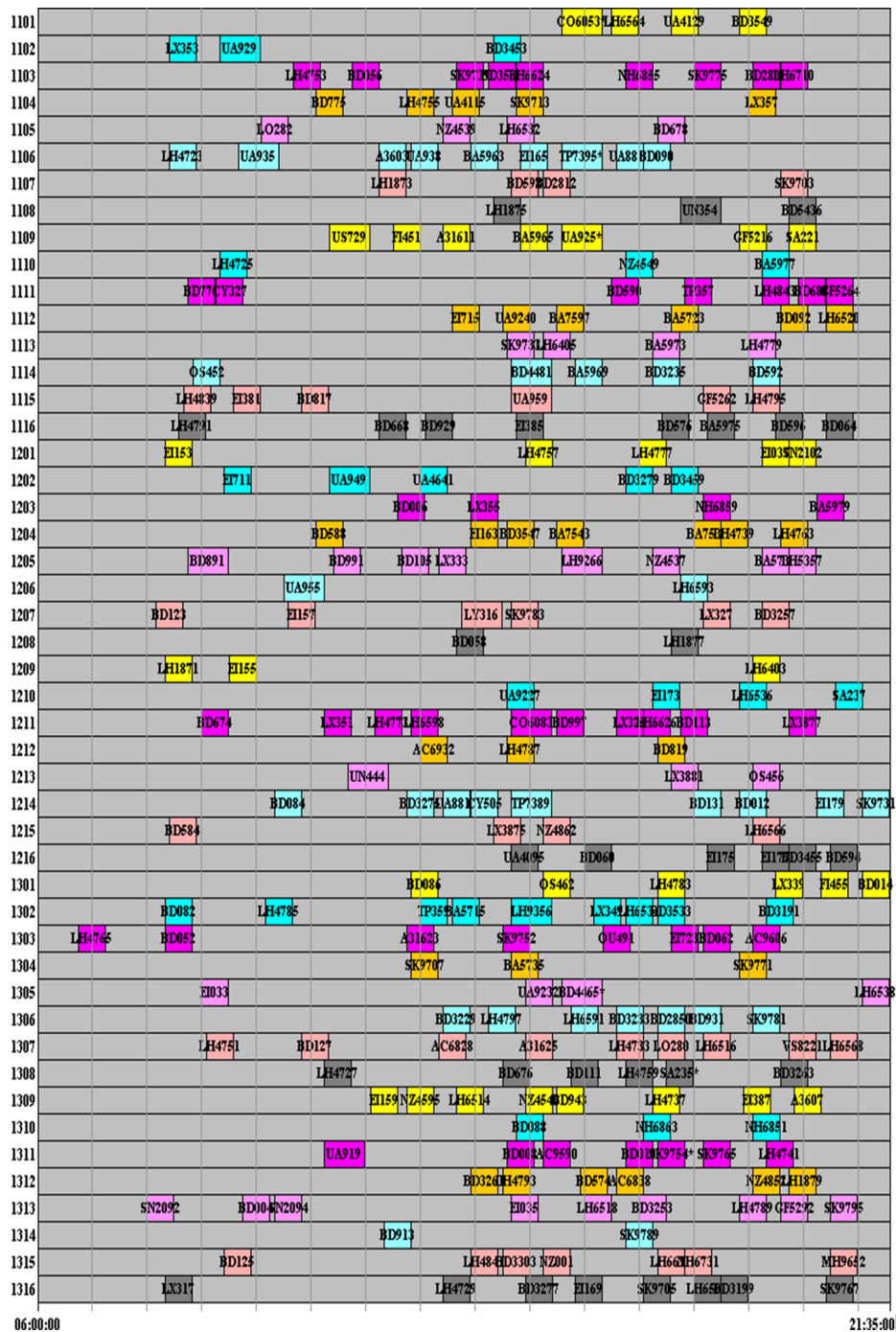


Fig. 1 Random allocation to stands of 270 flights from 1st March 2010.

This paper is structured as follows: Firstly, the problem description and model are presented, followed by a description of the considered algorithms. The results from applying the presented algorithms to the problem are then provided and various observations are made and explanations given. Finally some conclusions are presented in Section 5.

2 Problem Description and Model

The problem which is considered in this paper can be summarised as the assignment of available baggage sorting stations to already scheduled flights. For the baggage sorting station allocation problem, the flights will have already been assigned to stands, which are often grouped along piers around the terminals.

2.1 Airport Layout

The airport geometry plays an important role in the allocation of resources and the safety of the airport operations. An overview of the airport configurations and technologies for the transportation of passengers and baggage was presented by Pitt et al (2002) who concentrated on airport configurations and the availability of different types of resources. Rijsenbrij and Ottjes (2007) provided an overview of different elements of the baggage handling system and gave a description of the way in which baggage is currently handled, identifying potential areas of improvement.

Figure 2 provides a stylised diagram of an example layout. The stands are grouped on piers, which contain their baggage sorting stations at their base, placed perpendicularly to the pier. For any stand, it will be better to allocate the luggage to the sorting stations on the same side of the same pier. Alternatively, more distant sorting stations could be used, but these are less preferable. A ‘cost’ can be associated for a stand-sorting station pairing and one aim is to reduce this cost, by allocating as many flights as possible to their preferred sorting stations.

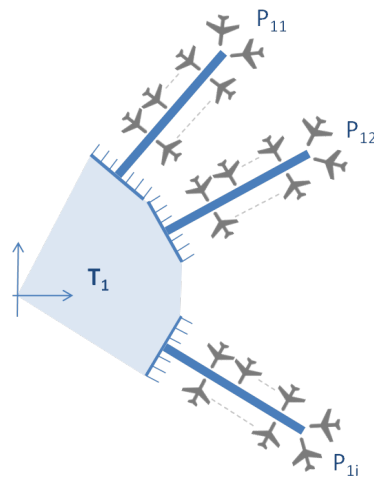


Fig. 2 Airport Layout.

The model which is utilised in this paper is appropriate for any airport where there are groupings of aircraft/gates which enforce a sorting station group preference (such as when aircraft are on piers) and where there is a distance or cost metric for the allocation of a sorting station to a flight, rather than being specific to the example layout described above. For example, at some airports, the sorting stations may be between the gates, in which case the preference for the distance/cost for allocating flight-sorting station pairs may be much stronger, but the group/pier preference may not be so strong.

2.2 Input Data and Constants

Table 1 summarises the various constants which are used in the model described in this paper. The relationship between the timing values is illustrated in Figure 3. Given these definitions, the service time of a flight is the duration from s_j to e_j and the ideal service time is the duration from t_j to e_j .

Table 1 List of the constants and input values for the model.

Name	Description
n	The total number of baggage sorting stations under consideration.
m	The total number of flights to which sorting stations should be allocated.
T_j	The base service time for flight j .
B_j	The desired buffer time for flight j .
R_j	The maximum reduction of service time allowed for flight j (we assume $R_j = B_j$ for this paper, so that the buffer time can be reduced but the base service time cannot).
e_j	The end service time for flight j .
t_j	The target starting service time for flight j , $t_j = e_j - T_j - B_j$, assuming the full buffer time is available.
C_j	A flight specific constant representing the amount of baggage to be processed for flight j . This determines the difficulty involved with allocating the flight to a sorting station which is further away. For example, this may represent the number of delivery trips required to move the baggage from the sorting station to the aircraft. In the absence of baggage load figures, we set $C_j = 1$ for all aircraft for this paper.
d_{ij}	The distance between baggage sorting station i and flight j .

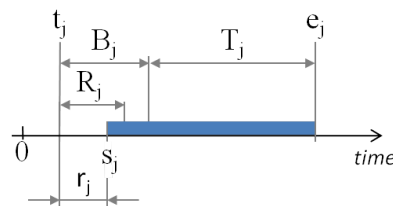


Fig. 3 View of the components of the service time.

2.3 Service Period

A service period is associated with each departing flight, during which the baggage for the flight is accumulated at the assigned baggage sorting station and finally loaded onto baggage carts for transfer to the flight. This service period may (optionally) be extended by applying a buffer time to prefer having a gap between the servicing of consecutive flights by the same sorting station.

2.4 Buffer Time

The buffer time is applied between two consecutive flights on the same baggage sorting station in order to absorb small disturbances in the real system behaviour. Buffer times are a common means of increasing robustness to avoid delays, as studied by Nikulin (2006) and Mulvey et al (1995). Buffer times were used in the scheduling of baggage sorting stations by Abdelghany et al (2006), and Wu and Caves (2004) used them in the optimization of the aircraft turnaround process. The Airport Gate Assignment Problem (GAP) has some similar characteristics to the baggage sorting station allocation problem and buffer times have been considered for the GAP by Hassounah and Steuart (1993), Yan and Chang (1997), Bolat (2000), Yan et al (2002) and Wu and Caves (2004). Yan and Huo (2001) and Yan et al (2002) analysed the effect and sensitivity of the buffer time, and Wei and Liu (2009) compared three distribution functions of buffer time with fixed buffer times using Fuzzy models. Buffer times have also been considered in other areas, such as the Resource Constrained Project Schedule Problem (RCPSP), where Long and Ohsato (2008) applied Fuzzy Critical Chain methods with buffer times.

Two constant ideal buffer time values have been considered in this paper, depending upon the flight destination type, namely European (short haul) or long distance, but buffer times could be introduced which depend upon individual aircraft types or even upon individual airline operators, routes or baggage handlers.

Given that the intention of using a buffer time is to reduce the impact from small perturbations in previous flight allocations to the same baggage sorting station, the buffer time has been allocated solely to the starting time of the service in this paper. If a previous flight closes its sorting station late due to delays, no re-scheduling will be required until all of the buffer time allocated to the next flight has been consumed by the delay. Of course, a flight could potentially consume the buffer on both sides of its service period, potentially opening a sorting station early and consuming the buffer time allocated to that flight, then closing the sorting station late and consuming the buffer time for the following aircraft.

2.5 Decision Variables

Table 2 lists the decision variables which are used in this model. The solution algorithms will attempt to find values of y_{ij} and r_j such that the constraints in Section 2.6 are met and the relevant objectives (e.g. maximising the allocations and minimising the reductions in service times) in Section 2.7 are improved.

Table 2 List of the decision variables which are used in this model.

Name	Description
y_{ij}	Specifies the allocation of flights to sorting stations. $y_{ij} = 1$ if baggage sorting station $i \in \{1, \dots, n\}$ is allocated to flight $j \in \{1, \dots, m\}$, and 0 otherwise.
r_j	Specifies the necessary reduction in service time for flight $j \in \{1, \dots, m\}$, given the allocated starting service time, s_j .
s_j	The allocated starting service time for flight $j \in \{1, \dots, m\}$, given that a sorting station can only service one flight at a time. s_j can be determined from r_j since $s_j = t_j + r_j$.

2.6 Constraints

Various constraints apply to the allocation of baggage sorting stations and can be summarised as follows.

2.6.1 Assignment Limits

Each flight must be assigned to at most one baggage sorting station, as expressed by Inequality (1). In normal operations, each flight should be allocated to exactly one sorting station, in which case Inequality (1) would instead be an equality. However, in extreme situations, where there are insufficient sorting stations (as considered in this paper) there may be no feasible allocation of flights to sorting stations such that all flights can be allocated, hence the inequality.

$$\sum_{i=1}^n y_{ij} \leq 1 \quad \forall j \in \{1, \dots, m\} \quad (1)$$

2.6.2 Reduction in Service

Baggage sorting stations can only be simultaneously used by one flight, so it may be necessary to reduce the service times of flights (usually by reducing the buffer times between flights) in order to allocate flights to the same sorting station. The principal objective is usually to maximise the assignment of baggage sorting stations to flights, as expressed by Formula (4).

For any pair of different flights j and l whose service times overlap (where $e_j \leq e_l$), if the overlap in service times is greater than the maximum allowed reduction R_l , then both flights cannot be assigned to the same baggage sorting station. Otherwise they may be assigned to the same baggage sorting station as long as the service time of flight l is reduced sufficiently to remove the overlap.

$$\begin{aligned} \forall j, l \in \{1, \dots, m\}, j \neq l, \text{ s.t. } e_j < e_l \text{ and } t_l < e_j, \forall i \in \{1, \dots, n\} : \\ r_l \geq (y_{ij} + y_{il} - 1) * (e_j - t_l) & \quad \text{if } (e_j - t_l) \leq R_l \\ y_{ij} + y_{il} \leq 1 & \quad \text{otherwise} \end{aligned} \quad (2)$$

2.6.3 Limit Service Reduction

The service time reduction may not exceed a limit, as expressed by Inequality (3).

$$0 \leq r_j \leq R_j \quad \forall j \in \{1, \dots, m\} \quad (3)$$

2.7 Objectives

A number of objectives need to be considered for this problem, and there is a trade-off to be made between them.

2.7.1 Maximise Assignment of Baggage Sorting Stations

The first and most important objective is to maximise the number of flights assigned to baggage sorting stations, as expressed by Formula (4). In practice at airports, this objective would probably be a hard constraint at most times, since all flights would normally have to be serviced, but we wish to observe the performance of the algorithms when there are too few sorting stations as well as when these are sufficient or plentiful.

$$\max \sum_{i=1}^n \sum_{j=1}^m y_{ij} \quad (4)$$

2.7.2 Minimise Distance

The distance between the assigned baggage sorting stations and the flights should be as short as possible. This objective aims to minimise the inconvenience, work and time involved in getting baggage to the aircraft, as previously discussed and could relate to preferences rather than strictly to distances, as discussed later. It can be expressed by Formula (5) where $\sum_{i=1}^n (y_{ij} * d_{ij})$ corresponds to the distance between flight j and its allocated baggage sorting station.

$$\min \sum_{j=1}^m \left(C_j * \sum_{i=1}^n (y_{ij} * d_{ij}) \right) \quad (5)$$

2.7.3 Fair Workload

The fairness objective corresponds to the minimisation of the total deviation of the actual usage of each baggage sorting station from the mean usage of all baggage sorting stations. This is expressed by Formula (6), where $e_j - s_j$ corresponds to the actual service time for the flight j , which is the usage time of the baggage sorting station. This objective aims to find a fairer allocation across sorting stations, as discussed in Abdelghany et al (2006).

$$\min \sum_{i=1}^n \left| \sum_{j=1}^m (y_{ij} * (e_j - s_j)) - \frac{\sum_{i=1}^n \sum_{j=1}^m (y_{ij} * (e_j - s_j))}{n} \right| \quad (6)$$

2.7.4 Minimise Reduction in Service

Given the detrimental effects that the reduction in service time has for the robustness of the allocation against real-life delays, it is advisable to minimise the total reduction in service time, thus maximising total buffer times. This objective can be expressed by Formula (7).

$$\min \sum_{j=1}^m r_j \quad (7)$$

2.7.5 Preferred Piers

Flights may have preferred piers and these should be considered when assigning baggage sorting stations. For this paper, we assume that it is preferable to allocate to each flight, sorting stations which are on the same pier. This objective is correlated to the distance minimisation objective, Formula (5) and is, therefore, not considered separately. The assignment of baggage sorting stations to preferred piers is considered in different ways by the different sorting station allocation algorithms which are described in Section 3.2.1, which differ in whether they first consider pier preferences or avoiding buffer time reductions.

2.7.6 Flights to the Same Destination

It is preferable that flights from the same carrier to the same destination be assigned to the same baggage sorting station, so that, for example, any delayed baggage could be transported on the next flight. However, flights would also normally be allocated to stands according to carrier, and potentially according to destination (or at least long-haul vs short-haul). Since we have utilised random stand allocation (see Section 4.1) for the experiments in this paper, it does not make sense to consider this objective and so it has been ignored.

2.7.7 Other Objectives

Other objectives are also possible, such as a reduction in the number of open sorting stations (to reduce the number of baggage handlers required). However, these are in direct conflict with equity and reduction in service considerations. These are not considered in this paper for reasons of space, although there are some observations made about these in Section 3.2.2, in the Last In First Out (LIFO) discussion.

A non-linear cost for service time reduction could also be used, so that fewer larger reductions are penalised greater than many lower reductions, since large reductions in the buffer time are far less favourable than smaller reductions. This will be an interesting area for future research but is not considered in this paper.

3 Algorithms

The constructive algorithms considered in this paper allocate baggage sorting stations to flights one at a time until no more allocations are possible. Flights are first ordered according to one of the considered flight ordering methods. A sorting station is then

selected for each in turn. The flight ordering and baggage sorting station allocation problems are considered below.

3.1 Flight Ordering Methods

The Flight Ordering method determines the order in which flights are selected for assignment. The following different sorting approaches are considered here:

1. **Order by Starting Time (OST)**. From the algorithm pseudo code presented therein, this appears to be what was previously used in Abdelghany et al (2006) and orders flights into ascending order of their t_j values.
2. **Order by Departure Time (ODT)**. This was previously used by Ding et al (2005) for the Airport Gate Assignment Problem (GAP). This orders flights into ascending order of their e_j values. When two flights have the same service end times, they are sorted by their target starting time t_j . When service time reductions are not permitted, sorting by service end times provides maximum allocations when using Last In First Out (LIFO) baggage sorting station selection and not constraining the set of sorting stations from which to select (see Section 3.2.2).

Order by Departure Time Lookahead and Improvement (ODTLI). Haralick and Elliott (1980) considered the concept of “Lookahead and anticipate the future in order to succeed in the present” and “Lookahead to the future in order not to worry about the past”. A type of look-ahead was also used in Voß et al (2005). The ODT flight ordering method could potentially perform badly on the objectives other than the maximisation of allocations and, in particular, the reuse of sorting stations can lead to an extremely inequitable allocation across sorting stations. The aim of the ODTLI look-ahead is to keep the ODT flight ordering but to look ahead when allocating sorting stations, to improve one of the other objectives while maintaining the maximal allocation of flights to sorting stations. The developed ODTLI algorithm looks ahead to find an allocation of future flights to sorting stations according to the ODT ordering, then attempts to find an improved allocation using the current Baggage Sorting Station Selection (BSSS) method, by exchanging some of the baggage sorting station allocations (using one-for-one swaps) without reducing the number of flights which are allocated to sorting stations.

3.2 Baggage Sorting Station Allocation

Once the flight to allocate has been identified, the next stage is to determine which sorting station to allocate it to. Baggage Sorting Station Allocation involves two stages. The first decision is upon which sets of baggage sorting stations to consider for allocation and in what order. In particular, whether only those for the same pier should be considered first, and whether service time reductions should be considered. The second decision involves the ranking of baggage sorting stations within each set, to enable the selection of an individual baggage sorting station to allocate.

3.2.1 Baggage Sorting Station Allocation Algorithms

The Baggage Sorting Station Allocation Algorithm determines which sets of baggage sorting stations (for example only those on the same pier, or on all piers) are consid-

ered, in which order, and at what point reduction in service is considered within each set. The baggage sorting stations within each set are then considered according to a selection priority given in Section 3.2.2. Algorithms A to E express different priorities. Algorithm A will attempt to allocate all aircraft to their own piers before considering allocating any aircraft to other piers, and Algorithm B will attempt to allocate all aircraft anywhere without reductions in service prior to reducing the service of any aircraft. Algorithms C and D are similar to A and B (respectively) but consider alternative piers or reductions in service for the current aircraft prior to considering the next aircraft, giving a much weaker preference overall. Algorithm E will not impose any restriction.

Algorithm A: Baggage Sorting Station Allocation Algorithm *A*

Order all the flights based on the current flight ordering algorithm, Section 3.1.
forall the flights do
 Consider sorting stations on own pier, without reduced service times.
 Otherwise, consider sorting stations on own pier allowing reduced service.
end
forall the unassigned flights do
 Consider any sorting station in airport, without reduced service times.
 Otherwise, consider any sorting station in airport, allowing reduced service.
end

Algorithm B: Baggage Sorting Station Allocation Algorithm *B*

Order all the flights based on the current flight ordering algorithm, Section 3.1.
forall the flights do
 Consider sorting stations on own pier, without reduced service times.
 Otherwise, consider any sorting station in airport, without reduced service.
end
forall the unassigned flights do
 Consider sorting stations on own pier allowing reduced service times.
 Otherwise, consider any sorting station in airport, allowing reduced service.
end

Algorithm C: Baggage Sorting Station Allocation Algorithm *C*

Order all the flights based on the current flight ordering algorithm, Section 3.1.
forall the flights do
 Consider sorting stations on own pier, without reduced service times.
 Otherwise, consider sorting stations on own pier allowing reduced service.
 Otherwise, consider any sorting station in airport, without reduced service.
 Otherwise, consider any sorting station in airport, allowing reduced service.
end

Algorithm D: Baggage Sorting Station Allocation Algorithm *D*

Order all the flights based on the current flight ordering algorithm, Section 3.1.
forall the flights do
 Consider sorting stations on own pier, without reduced service times.
 Otherwise, consider any sorting station in airport, without reduced service.
 Otherwise, consider sorting stations on own pier allowing reduced service.
 Otherwise, consider any sorting station in airport, allowing reduced service.
end

Algorithm E: Baggage Sorting Station Allocation Algorithm *E*

Order all the flights based on the current flight ordering algorithm, Section 3.1.
forall the flights do
 Consider any sorting station in airport, without reduced service times.
 Otherwise, consider any sorting station in airport, allowing reduced service.
end

In each case, once the algorithm has determined the set of sorting stations to consider, the appropriate sorting station to allocate is determined by the baggage sorting station selection method which is being used (see Section 3.2.2).

For this paper, we have concentrated on baggage sorting station allocation algorithms A, C and E since these consider the effect of prioritising the pier or not. Future work will consider algorithms B and D, which prioritise avoiding service time reductions. These have been included here for completeness.

3.2.2 Baggage Sorting Station Selections

The Baggage Sorting Station Selection method determines which of the baggage sorting stations in the current set should be assigned to the current flight. The following methods are considered:

1. **First In First Out (FIFO):** The baggage sorting station with the earliest free service time amongst all of the baggage sorting stations in the set under consideration is selected. This will initially keep opening new service stations, while they exist, since a new one would always be least recently used. This is useful for meeting the fairness objective expressed by Formula (6).
2. **Last In First Out (LIFO):** The most recently used baggage sorting station amongst those in the set being considered is selected. This selection reduces the number of baggage sorting stations in use at any time, since a new baggage sorting station is only opened when the previous ones cannot be assigned to the flight. With the ODT and ODTLI flight ordering methods, no reduction in service time and allocation algorithm E (so that all sorting stations are considered, rather than only the ones on the preferred pier), this selection method guarantees the maximum allocations (maximising the objective expressed by Formula (4)), by minimising the wasted/idle time between flights, Ding et al (2004) and Cormen et al (2001).
3. **Closest:** The baggage sorting station with the smallest distance to the current flight is selected from those in the set under consideration. This considers both new and previously used service stations. This method is useful for meeting the distance reduction objective expressed by Formula (5).

4 Results

This section details the experiments which were performed to evaluate the differences between the algorithms which were described in Section 3, with differing numbers of sorting stations available for allocation to flights. The behaviour can, therefore, be studied when there are too few sorting stations as well as when the sorting stations are plentiful.

4.1 Problem Data

Since it would be unrealistic to assume that baggage from a flight at a stand in a terminal is serviced by a baggage sorting station in another terminal (e.g. passengers usually go through security and board flights from the same terminal where they checked in their baggage), it was decided to centre the analysis on a single terminal. In order to obtain a more realistic spread of flights (since flights are rarely evenly spread over the day), publicly available flight time data from the British Airports Authority (BAA) website was used for these experiments, consisting of 219 flights from the 16th December 2009 and 270 flights from the 1st March 2010, all departing from London Heathrow Airport. The data for the 16th December has a large number of flights clustered around the middle of the day, whereas the data for the 1st March has flights which are more spread out over a long period. The 1st March is a more normal day, but the clustering of the 16th December is extremely useful for the purposes of this paper, where we wish to consider the effects under very resource constrained conditions.

Since the data did not contain the assignment of stands to flights, different problems were generated by allocating the flights randomly to the stands avoiding any overlap on a single stand. In order to minimise any introduced bias from these random allocations, a hundred different random allocations were generated. We realise, of course, that real schedules will have some bias, for airlines preferences, and future work will consider the real stand allocations. One example allocation is illustrated by Figure 1. The box-and-whisker diagrams in the following sections (from Figure 6 onwards) illustrate the results of the sorting station allocation algorithms (which are themselves deterministic) across the 100 different stand allocations. In each diagram, results are shown next to each other for each number of baggage sorting stations and are in the same order as they are listed in the key. In allocating flights to stands, it has been assumed that all of the stands were suitable for any of the aircraft. The available stands were assumed to be equally distributed over three piers, with 16 stands per pier.

Various experiments were executed using a single threaded Java application, 14, 16, 18, 20, 22, 24, 26, 28, 30, 34 and 36 baggage sorting stations per pier, changing the total number of sorting stations in increments of 6. For the purpose of the distance reduction objective, a distance of one unit was assumed between different sides of a pier and a distance of two units was assumed between different piers (see Figure 2), so that it is preferable to use the other side of the same pier before considering sorting stations for other piers. For the moment, we consider that reductions in service time can only reduce the buffer time rather than the base service time, so $R_j = B_j$. Service times were set so that $T_j = 1$ hour and $B_j = 15$ minutes for European flights, and $T_j = 1\frac{3}{4}$ hours and $B_j = 30$ minutes for non European (longer haul) flights, since these are usually larger flights with more baggage and requirements to check in earlier.

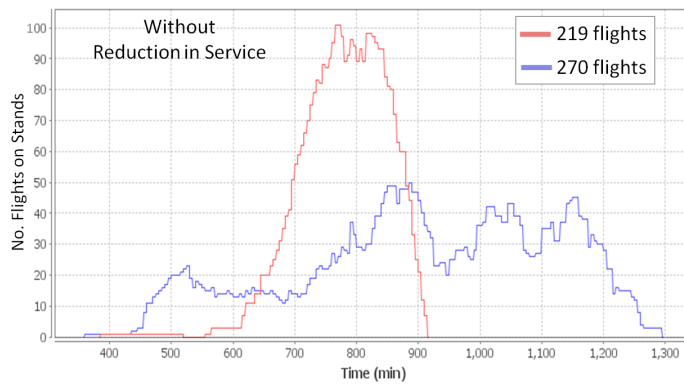


Fig. 4 Without Reduction in Service (including buffer times)

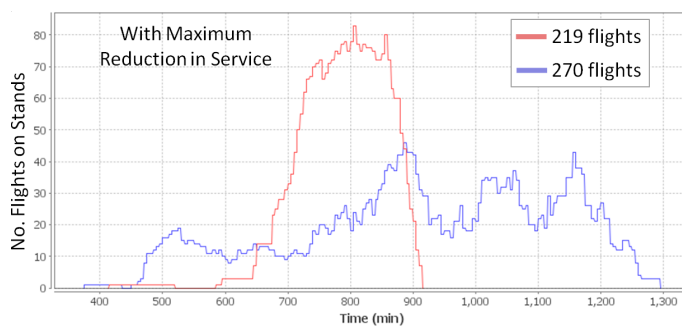


Fig. 5 With Maximum Reduction in Service (i.e. no buffer times)

Figures 4 and 5 show the total number of flights which required service at different times of the day, with and without the buffer times. Figure 4 shows the number of sorting stations which are needed when full buffer times are used (i.e. where there is no service time reduction allowed) and Figure 5 represents the number of flights actually requiring service at that time (i.e. no buffer times are included). It is possible to draw the following conclusions:

1. The number of sorting stations required differs greatly over the day.
2. With a limited number of baggage sorting stations, the maximum heights of the lines in Figures 4 and 5 can be considered to be an indication of the difficulty of the allocation problem.
3. The 219 flight dataset is more concentrated, requiring far more sorting stations over a shorter time period.
4. Fewer sorting stations are required at the peaks when buffer times are not included, but the absence of the buffer times would result in less robust solutions.

Various experiments were executed using a single threaded Java application, running on a 3GHz Intel(R) Core(TM)2 Duo CPU, desktop with 2GB RAM, running Windows XP (SP3). Each execution of the constructive algorithm took no more than 9 milliseconds.

4.2 Optimal allocations

As discussed in Section 3.2.2, Order by Departure Time (ODT) with, Last In First Out (LIFO) sorting station selection, no reductions in service permitted and consideration of all sorting stations at once (Baggage Sorting Station Allocation Algorithm (BSSAA) E) provides optimal allocations.

The same problem was formulated as an Integer Linear Programme and solved using CPLEX. Unsurprisingly, CPLEX provided the same (optimal) number of allocations. However, it took a hundred times as long to solve.

We wished to investigate whether CPLEX could also find an optimal solution for a weighted sum of the different objectives. We generated a combined objective function using relative weights of 10 for the maximum assignment objective (expressed by Formula (4)), 0.005 for the minimising reduction in service objective (expressed by Formula (7)) and 1 for the distance minimisation objective (expressed by Formula (5)), so that these would be considered as primary, secondary and tertiary objectives, respectively, taking into account the relative magnitude of each component. Experiments were executed for the 219 and 270 flight cases using CPLEX but the program was stopped after 8 hours with no improvement, if it had not run out of memory by that point (having consumed up to 87GB of virtual memory). CPLEX was not able to achieve proven optimality in any of the cases and a lot of memory was utilised in each case.

4.3 Initial Observations

Experiments were executed for different numbers of baggage sorting stations, with each of the Baggage Sorting Station Allocation Algorithms (BSSAAs) and Baggage Sorting Station Selection (BSSS) methods. Two cases were considered: without reduction in service times (i.e. requiring full buffer times) and with reduction in service times (i.e. allowing buffer times to be reduced).

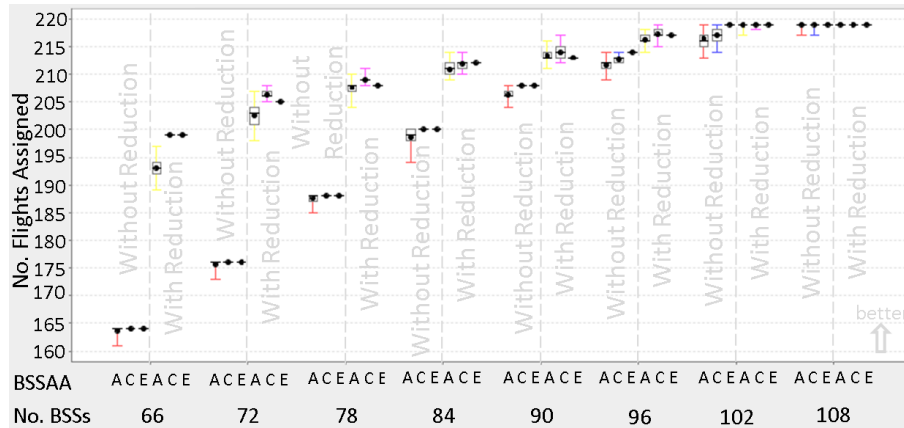


Fig. 6 Number of sorting stations assigned for 219 flights for ODTLI flight ordering and LIFO sorting station selection, with and without permitting reductions in service.

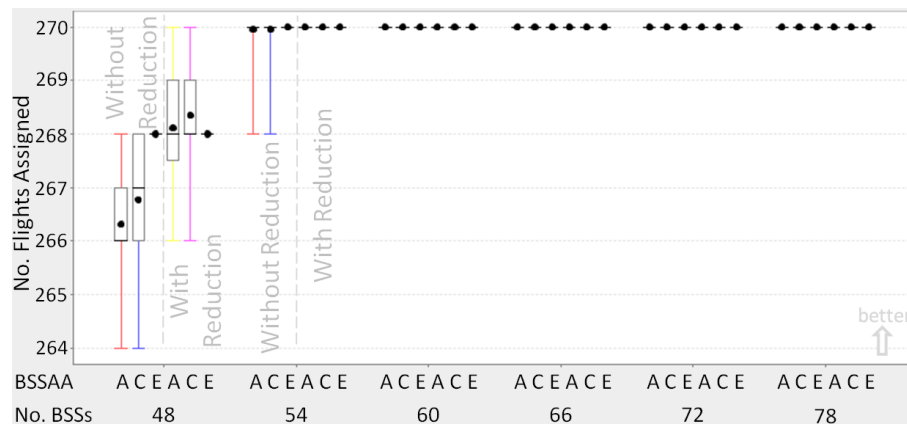


Fig. 7 Number of sorting stations assigned for 270 flights for ODTLI flight ordering and LIFO sorting station selection, with and without permitting reductions in service.

Figures 6 and 7 show (for the 219 and 270 flight problems, respectively) the number of sorting stations which could be allocated to flights for the Order by Departure Time Lookahead and Improvement (ODTLI) flight ordering with LIFO sorting station selection, and both when no reduction in service is allowed and when reduction in service is allowed for various numbers of available sorting stations.

These figures give an idea of the behaviour of the allocation as the flight density and number of baggage sorting stations change. The following can be observed:

1. As expected, more assignments are achieved when allowing reductions in service time, as long as there are insufficient sorting stations available. Maximising allocations is probably the most important objective.
2. When service time reductions are permitted, the ODTLI flight ordering, with LIFO sorting station selection and allocation algorithm E no longer guarantees the maximum allocation of sorting stations. Algorithms A and C, which consider reductions in service prior to considering other piers (thus reducing service times more often), sometimes achieve better allocations (e.g. 72 and 78 baggage sorting stations Figure 6 and with 48 sorting stations Figure 7).

In order to determine the maximum sorting station allocations with reduction in service times permitted, experiments were executed with the buffer times removed (equivalent to maximal service time reduction), ODTLI flight ordering, sorting station allocation algorithm E and LIFO sorting station selection. The results are shown in Figures 8 and 9, for the 219 and 270 flight problems, respectively. More flights can be allocated when reductions are permitted, as expected, until sufficient sorting stations are available to allocate the maximum allocation even without needing reductions. In most cases, allowing reduction is almost as good as using maximum reductions.

The results for maximal reductions can be compared to the earlier results where reductions were and were not allowed. With maximum reductions (i.e. no buffer times), the maximum allocation occurs when there are 84 and 48 baggage sorting stations for the 219 and 270 flight problems respectively. Without reductions, the maximum allocation occurs when there are 102 and 54 baggage sorting stations for the 219 and 270 flight problems respectively. These values are surprisingly close to the lower bounds

for the allocations which were given in Figures 4 and 5, which were 83, 46, 101 and 50 for these four cases, respectively. In fact, given that sorting stations were increased in multiples of 6 (2 extra at a time per pier), these are the lowest values considered which met the lower bound.

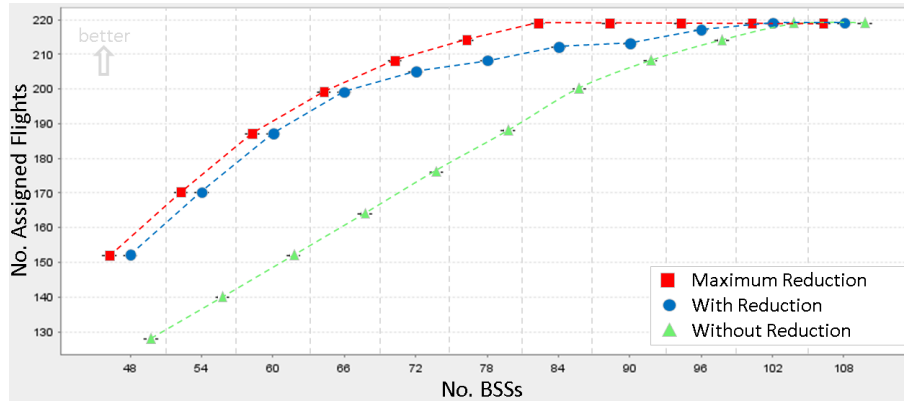


Fig. 8 Number of assignments for 219 flights for ODTLI flight ordering, sorting station allocation algorithm E with LIFO selection, contrasting the results for maximum reductions against the normal behaviour with and without permitting reductions.

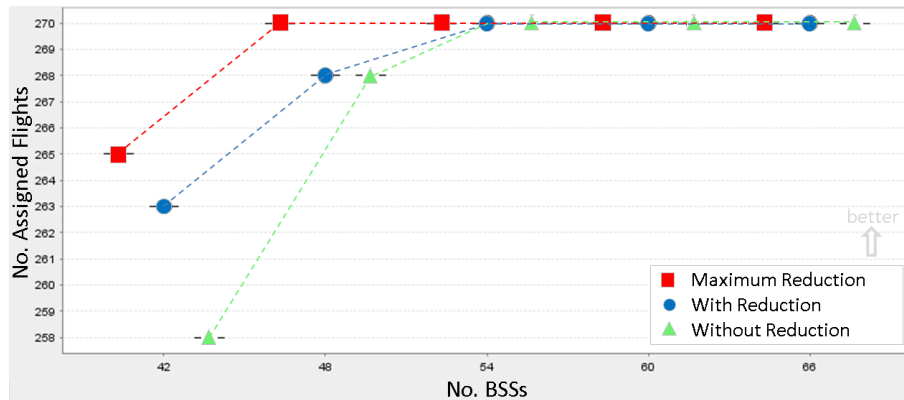


Fig. 9 Number of assignments for 270 flights for ODTLI flight ordering, sorting station allocation algorithm E with LIFO selection, contrasting the results for maximum reductions against the normal behaviour with and without permitting reductions.

Given the maximum allocation results and the obvious benefits of allowing service time reductions, the following analysis will consider only the case where reductions in service are permitted.

4.4 Comparison of Allocations With Service Reduction

Figures 10 and 11 show the number of sorting station assignments for the ODTLI and Order by Starting Time (OST) flight ordering methods, with sorting station allocation algorithms A, C and E and LIFO selection.

Figure 10 shows that ODTLI flight ordering provides a better allocation when there are fewer sorting stations, but at some point, as the number of sorting stations increases, the difference decreases and as it approaches the number necessary for optimal allocation, the OST flight ordering actually improves upon ODTLI.

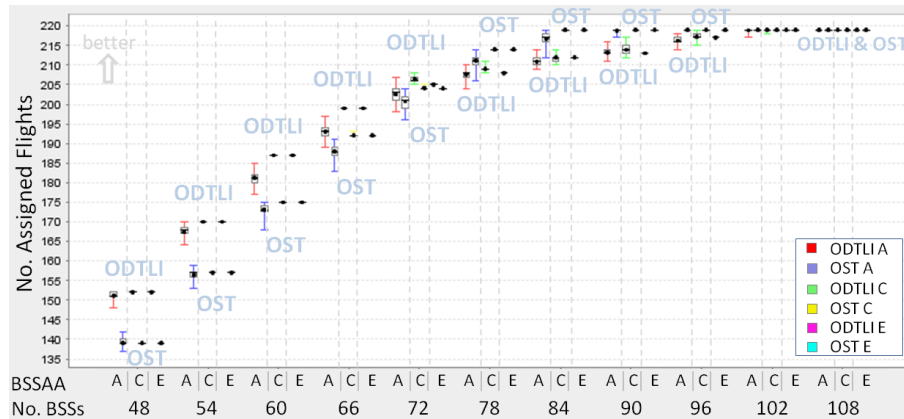


Fig. 10 Number of assignments for 219 flights for ODTLI and OST flight ordering methods, sorting station allocation algorithms A, C and E and LIFO sorting station selection, with service reduction.

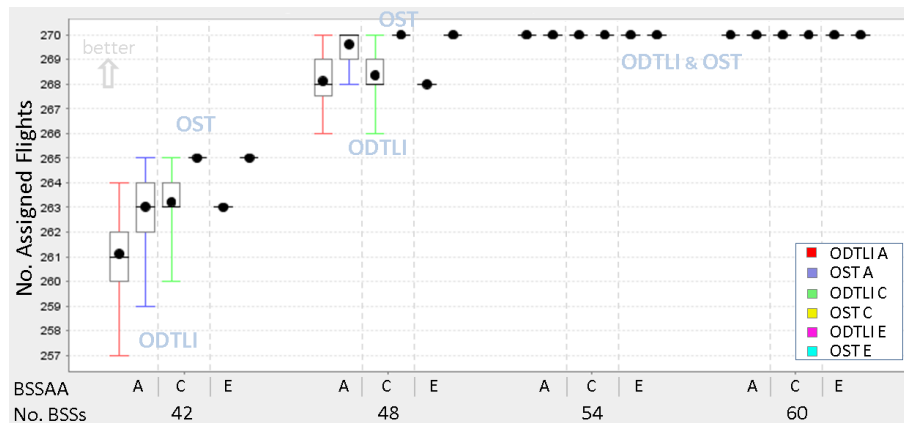


Fig. 11 Number of assignments for 270 flights for ODTLI and OST flight ordering methods, sorting station allocation algorithms A, C and E and LIFO sorting station selection, with service reduction.

For 219 flights, this point is 78 baggage sorting stations, which is close to the 84 required for maximum allocations when no buffer time is used. Since ODT and ODTLI order the flights by their departure time, where flights have similar service starting times, preference will be given to flights with shorter service times. On the other hand, the OST choice of flights could be considered to be preferring flights with longer service times (for similar departure/end of service times). Comparison of some resulting allocations showed that, perhaps counter-intuitively, not only was ODTLI failing to allocate more flights at these times, but the flights which were not allocated had longer service times than those which OST failed to allocate. Indeed, there were cases where every aircraft which OST failed to assign was a short-haul flight and every aircraft which ODTLI failed to assign was a long haul flight. The ordering appears to be important in this case. By allocating long-haul flights first, the OST algorithm is able to fit short-haul flights into the remaining gaps (with appropriate service time reductions). However, by allocating short-haul flights first the ODTLI is then unable to schedule the long-haul flights which remain, resulting in fewer allocations.

The key to understanding this behaviour is to consider the size of the remaining gaps. When there are few sorting stations, the ability of the ODTLI choice to minimise the gaps is a useful one and results in more sorting station allocations than the OST flight ordering. However, as the number of sorting stations increases, the remaining gaps start to become big enough to fit the short-haul aircraft into them and OST performs better.

Figure 11 is less interesting, since the situation was less restricted. Due to the lower density of the 270 aircraft dataset, Figure 11 effectively matches only the right hand side of Figure 10.

Further experiments showed that this behaviour was not restricted to the LIFO sorting station selection method, but also occurred for the FIFO and ‘Closest’ sorting station selection methods, and at the same number of sorting stations.

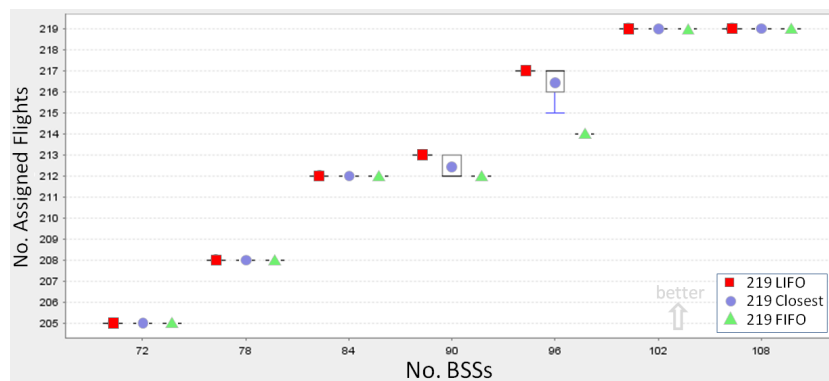


Fig. 12 Total number of sorting station assignments for 219 flights, for ODT flight ordering, sorting station allocation algorithm E, with service reduction.

Figure 12 shows the number of flights that were allocated to baggage sorting stations when ODT ordering of flights was considered and sorting station allocation algorithm E was used, with the ‘Closest’, First In First Out (FIFO) and LIFO sorting station selection methods, allowing a comparison to be made between the performances

of the sorting station selection methods. Results are shown only for 219 flights, since this is the more restricted case and the 270 flight case has the same structure. ODT was used in this case rather than ODTLI, to avoid any influence from the look-ahead element of the algorithm, but we note that ODTLI always performed at least as well as ODT for all of these examples. It can be observed that the LIFO selection method was the best in terms of the number of sorting stations allocated (it was always at least as good as the others), followed by the ‘Closest’ method (which was always at least as good as the FIFO method), with the FIFO method being the worst.

4.5 Comparison of Distances With Service Reduction

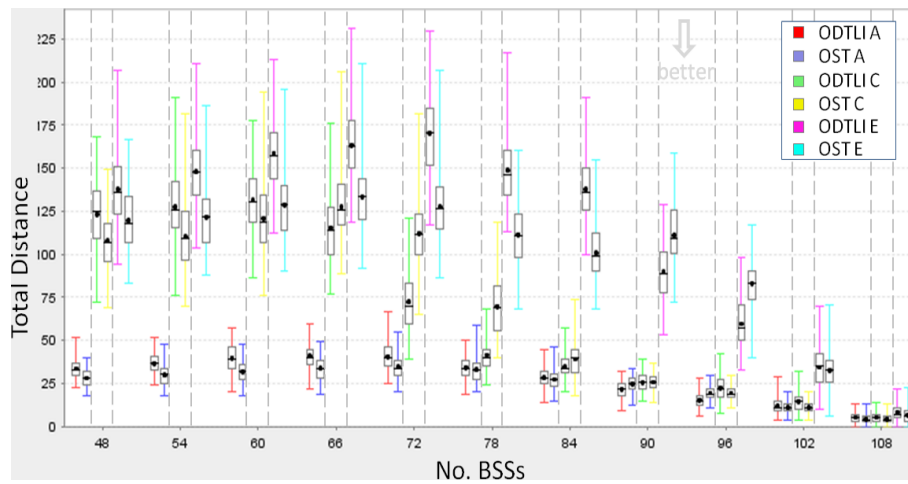


Fig. 13 Total distance results for 219 flights with ODTLI and OST flight ordering methods and ‘Closest’ sorting station selection, with service time reductions.

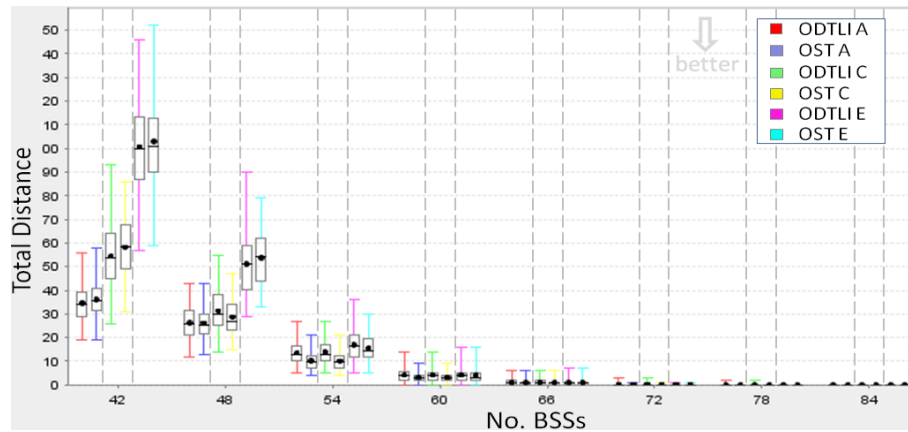


Fig. 14 Total distance results for 270 flights with ODTLI and OST flight ordering methods and ‘Closest’ sorting station selection, with service time reductions.

Figures 13 and 14 show the results as far as the distance reduction objective (expressed by Formula (5)) is concerned. These show the total distance between the assigned baggage sorting stations and the stand at which the flight is located. Results are shown for the three sorting station selection algorithms A, C and E, with ‘Closest’ sorting station selection and the ODTLI and OST flight ordering methods.

The distance basically measures the number of flights that could not be allocated to sorting stations on their preferred pier. It can be observed that the total distance decreases in all cases as the number of sorting stations is increased, since sorting stations become available on the preferred pier. Even after all flights can be allocated to sorting stations, the distances can be positive, since the availability of a sorting station at the terminal does not imply that it is on the correct pier for the flight.

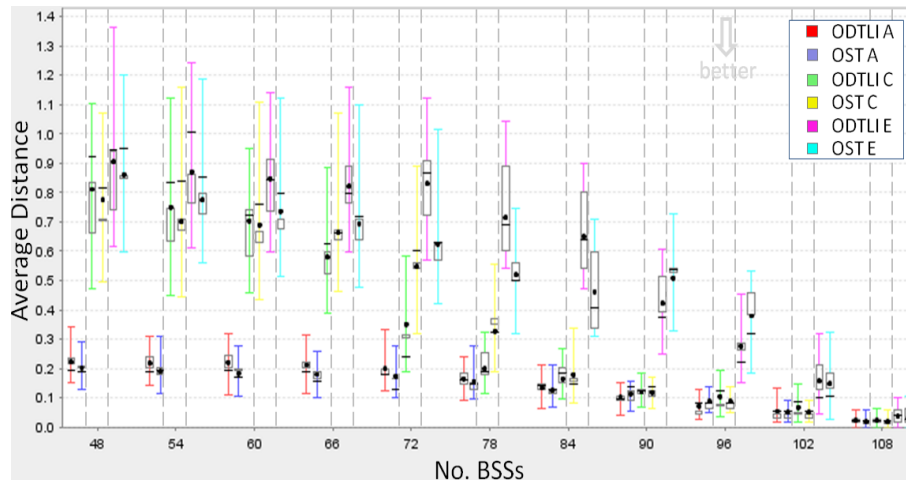


Fig. 15 Average distance for 219 flights with ODTLI and OST flight ordering methods and ‘Closest’ sorting station selection, with service reduction.

Algorithm A can be observed to perform better than algorithms C, and E when there is a shortage of piers. However, Algorithm E allocates more flights to sorting stations, and unallocated flights are not included, so some of the improvement could be due to the unallocated flights. Figure 15 shows the average distance per flight, to avoid this problem, and Algorithm A can be observed to also attain a lower average distance. This is expected since Algorithm A attempts to allocate to the same pier first and considers applying a service time reduction before considering other piers. For similar reasons, Algorithm C performs better than Algorithm E.

4.6 Fair Workload With Reduction in Service

It is also useful to consider the deviation of the total usage times of the sorting stations from the average usage time. This is the fairness objective which was expressed by Formula (6). Figures 16 and 17 show the total seconds deviation from the average usage across all baggage sorting stations, for allocation Algorithm E, comparing the

results for the ‘Closest’, FIFO and LIFO sorting station selection methods and the ODTLI and OST flight ordering methods.

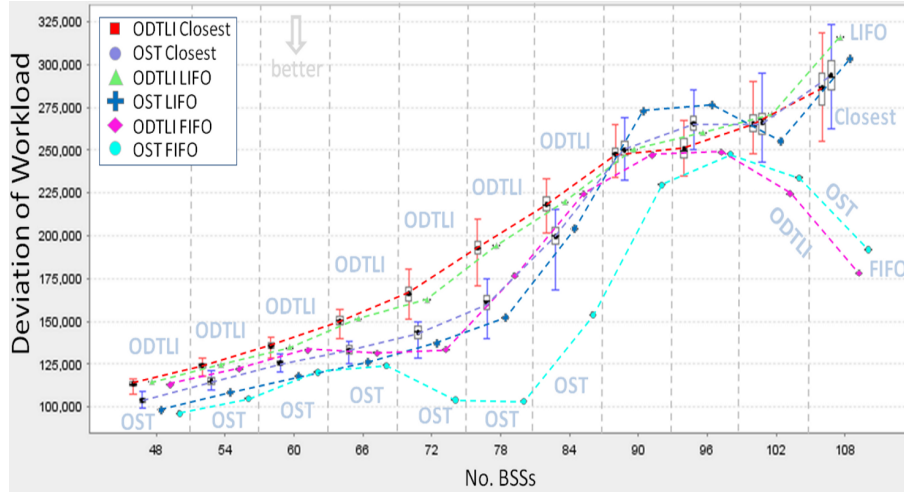


Fig. 16 Fair Workload for 219 flights with sorting station allocation algorithm E and reduction in service permitted.

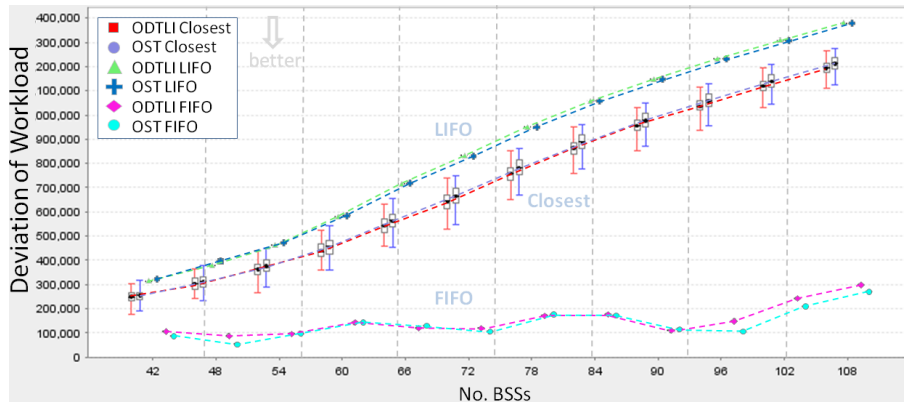


Fig. 17 Fair Workload for 270 flights with sorting station allocation algorithm E and reduction in service permitted.

The FIFO method can be considered to take this objective into account, and, indeed, consistently performs better than LIFO for both flight ordering methods.

Since the LIFO method will continue to re-use the same sorting stations where this is possible, increasing the number of sorting stations will further increase the inequity.

The FIFO method, on the other hand, will cycle through the sorting stations, giving a fairer allocation of flights to sorting stations. However, long-haul and short-haul flights are treated identically, resulting in differences in the total service times. These differences will depend upon how many of the long haul flight allocations coincide so that they are allocated to the same service stations, and a cyclic-type behaviour is observed as the number of sorting stations is increased.

4.7 Reduction in Service

Results for the reduction in service objective, expressed by Formula (7), are summarised in Figures 18 and 19 which represent the total reduction in service for all allocated flights, with differing numbers of baggage sorting stations, ODTLI and OST flight ordering and ‘Closest’, LIFO and FIFO baggage sorting station selection.

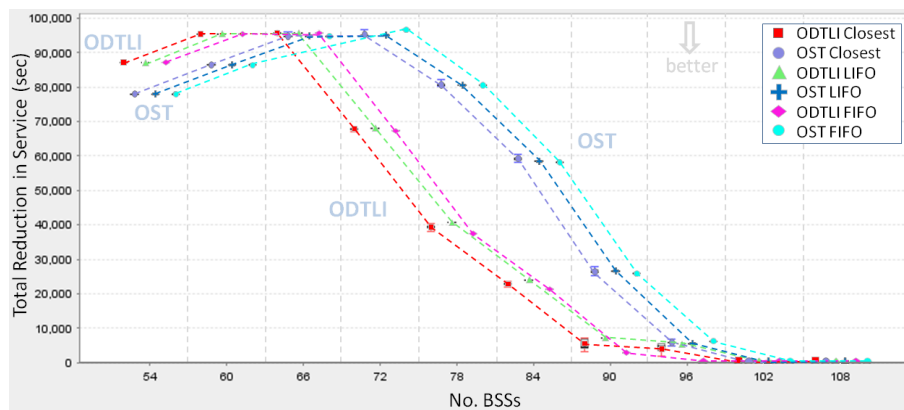


Fig. 18 Total reduction in service time for 219 flights with sorting station allocation algorithm E and reduction in service permitted.

As the number of sorting stations is increased, more of the allocations are achieved by means of the reduction in service. Thus, the seconds reduction in service increases. Eventually the point is reached where the number of sorting stations is sufficient to allow the allocations to be made with ever decreasing reductions in the service time, so the total reduction in service decreases, until eventually all flights can be allocated without any reductions in service time. Figure 20 shows the average reduction in service time for 219 flights, showing that the increase in the number of allocated flights is the origin of the initial increases in the time reduction in total service time as more sorting stations are added.

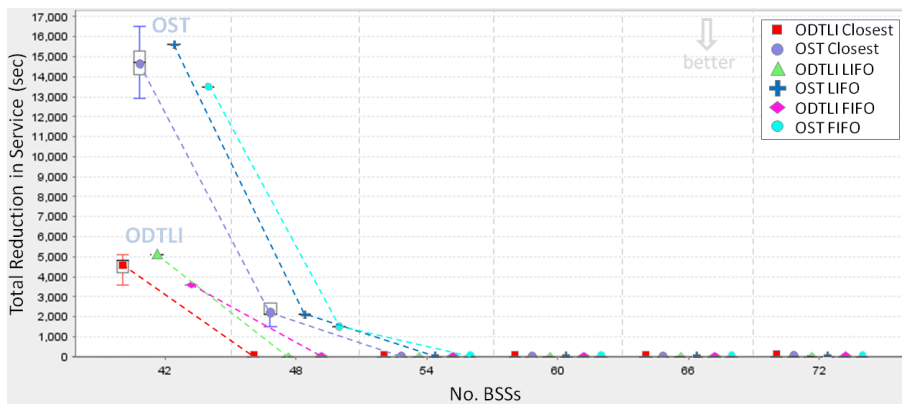


Fig. 19 Total reduction in service time for 270 flights with sorting station allocation algorithm E and reduction in service permitted.

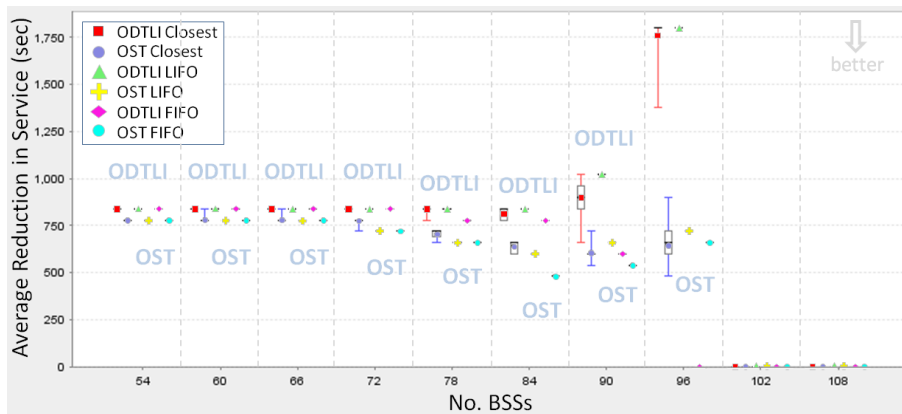


Fig. 20 Average Reduction in Services for 219 flights with sorting station allocation algorithm E and reduction in service permitted.

5 Conclusions

We have observed that (perhaps unsurprisingly) the flight density (the number of flights requiring service at any time in the day) is more important than the total number of flights when determining the number of sorting stations which are required throughout the day. We have also identified that the allocation methods (flight ordering, sorting station allocation algorithm and selection method) have different effects and can prefer different objectives. We have identified some points, in terms of the number of baggage sorting stations, where the performance of the algorithms changes and have noted that these depend upon the distribution of the flights over time. Finally, we have noted that the choice of whether or not to allow reductions in service time can affect the relative efficacy of the algorithms. In particular, if reductions in service time are to be permitted, then it may be better to select an algorithm which will not minimise the gap sizes, since these are then less likely to be usable by other flights after service

time reductions have been applied. Together, these effects show that the appropriate algorithm to be used depends not just upon the objective which is under consideration but also upon the problem characteristics and the relative flight density in relation to the number of sorting stations available.

The aim of this research was not to identify the perfect constructive algorithm, but to gain insights into the differing behaviour of the algorithms, particularly when service time reductions are permitted. We plan to utilise these insights in the development of a perturbative algorithm and potentially to utilise a number of these constructive algorithms to produce high quality initial solutions according to the different objectives which are under consideration. For example, their use with metaheuristic solution methods could potentially improve the performance of such methods (in terms of convergence time even if not solution quality) by providing higher quality initial solutions.

Acknowledgements

We are grateful to National Air Traffic Services Ltd (NATS) and EPSRC for providing funding for this project, and especially John Greenwood (NATS) for his continuous support.

References

- Abdelghany A, Abdelghany K, Narasimhan R (2006) Scheduling baggage-handling facilities in congested airports. *Journal of Air Transport Management* 12(2):76 – 81
- Air Transport Users Council (2009) Annual report 2008/09. Tech. rep., AUC
- Bolat A (2000) Procedures for providing robust gate assignments for arriving aircrafts. *European Journal of Operational Research* 120(1):63 – 80
- Cormen TH, Leiserson CE, Rivest RL, Stein C (eds) (2001) *Introduction to Algorithms*. The Massachusetts Institute of Technology Press
- Ding H, Lim A, Rodrigues B, Zhu Y (2004) Aircraft and gate scheduling optimization at airports. *Hawaii International Conference on System Sciences* 3:1530 – 1605
- Ding H, Rodrigues AL, Zhu Y (2005) The over-constrained airport gate assignment problem. *Computers and Operations Research* 32(7):1867 – 1880
- Federal Aviation Administration (2010) Faa aerospace forecast fiscal years 20102030 - forecast highlights. Tech. rep., Federal Aviation Administration, URL http://www.faa.gov/data_research/aviation/aerospace_forecasts/2010-2030/
- Haralick RM, Elliott GL (1980) Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14:263 – 313, URL http://www.cs.uu.nl/docs/vakken/cons/paper_adriano.pdf
- Hassounah MI, Steuart GN (1993) Demand for aircraft gates. *Transportation Research Record* 1423(1423):22 – 33, URL <http://pubsindex.trb.org/view.aspx?id=390306>
- ICAO (2010) Icao news release - pio 06/10. Tech. rep., International Civil Aviation Organization, URL <http://icaopressroom.files.wordpress.com/2010/07/pio-06-10-en.pdf>
- Long LD, Ohsato A (2008) Fuzzy critical chain method for project scheduling under resource constraints and uncertainty. *International Journal of Project Management* 26:688 – 698

- Mulvey JM, Vanderbei RJ, Zenios SA (1995) Robust optimization of large-scale systems. *Operational Research* 43:254 – 281, URL <http://coral.ie.lehigh.edu/ipseminar/robust/mulvey-et-a195.pdf>
- Nikulin Y (2006) Robustness in combinatorial optimization and scheduling theory: An extended annotated bibliography. URL http://www.optimization-online.org/DB_HTML/2004/11/995.html
- Pitt M, Wai FK, Teck PC (2002) Technology selection in airport passenger and baggage systems. *Facilities* 20(10):314 – 326, DOI 10.1108/02632770210442992
- Rijssenbrij JC, Ottjes JA (2007) New developments in airport baggage handling systems. *Transportation Planning and Technology* 30(4):417 – 430
- US Department of Transportation (2010) Air travel consumer report. Tech. rep., U.S. Department of Transportation, URL <http://airconsumer.dot.gov/reports/2010/April/2010AprilATCR.pdf>
- Voß S, Fink A, Duin C (2005) Looking ahead with the pilot method. *Annals of Operations Research* 136:285 – 302, URL http://www.smartframe.de/studentsatwork/literatur/sel_vossduin.pdf
- Wei D, Liu C (2009) Fuzzy model and optimization for airport gate assignment problem. In: *IEEE International Conference on Intelligent Computing and Intelligent Systems, ICIS, IEEE, Shanghai, vol 2*, pp 828 – 832, DOI 10.1109/ICICISYS.2009.5358268
- Wu CL, Caves RE (2004) Modelling and optimization of aircraft turnaround time at an airport. *Transportation Planning and Technology* 27(1):47 – 66, DOI 10.1080/0308106042000184454
- Yan S, Chang CM (1997) A network model for gate assignment. *Journal of Advanced Transportation* 32(2):176 – 189, DOI 10.1002/atr.5670320204
- Yan S, Huo CM (2001) Optimization of multiple objective gate assignments. *Transportation Research Part A: Policy and Practice* 35(5):413 – 432
- Yan S, Shieh CY, Chen M (2002) A simulation framework for evaluating airport gate assignments. *Transportation Research Part A: Policy and Practice* 36(10):885 – 898

SDP Relaxations for Two Stage OFDMA Networks with and without Probabilistic Constraints

Pablo Adasme · Abdel Lisser · Ismael Soto

Abstract In this paper, we propose a two stage stochastic binary quadratic program for OFDMA wireless networks. The goal is to minimize the total power consumption of the network subject to user bit rates, sub-carrier and modulation constraints. Then, we extend the quadratic model by introducing a knapsack probabilistic constrained approach from [2] on the user bit rate constraints of the second stage problem. In order to compute lower bounds, we derive linear and semidefinite programming relaxations (Resp. LP and SDP). Numerical results show tight lower bounds for the SDP relaxations when compared to the LP ones. Moreover, near optimal solutions are found by the SDP relaxation without probabilistic constraints.

1 Introduction

Wireless Networks (WNs) have become one of the most hot topic research fields in last decades (See for instance [3,20,12,27] and references therein). Recent technological developments such as smart phones and personal computers have increased the use of wireless networks due to its mobility and flexibility when establishing communication on real time. The IEEE 802.16 standards for wireless metropolitan area networks (WMANs) are examples of these new technologies which are being considered as potential candidates for fixed, portable and mobile air interface broadband wireless access networks. Although standards are remarkably improving, it all indicates that in the near future, more intelligent and powerful mobile equipments will be needed to satisfy the users increasing demands. However, since power and bandwidth channel capacities are scarce resources and no technologies are yet prepared to satisfy these future demands, it follows that most of research efforts are mainly focused on multi-carrier

Pablo Adasme - Abdel Lisser
Laboratoire de Recherche en Informatique, Université Paris-Sud XI, Bâtiment 490, 91405, Orsay Cedex France
E-mail: pablo.adasme@lri.fr, abdel.lisser@lri.fr

Ismael Soto
Departamento de Ingeniería Eléctrica, Universidad de Santiago de Chile, Avenida Ecuador 3519 Santiago, Chile
E-mail: ismael.soto@usach.cl

transmission methods. Among these transmission methods, orthogonal frequency division multiple access (OFDMA) appears as one of the most interesting candidates since it is based on OFDM technology which divides the frequency bandwidth channel into several orthogonal sub-carriers (or sub-channels). Besides, it offers an increased robustness to overcome multi-path distortions when compared to conventional single-carrier systems [20]. In order to efficiently assign the bandwidth channel capacity and power in an OFDMA network, several mathematical formulations have been proposed in the literature [3]. Although, from a technological point of view rather than taking into account the intrinsic combinatorial nature of the problem (see for instance [22,3,12,27]). For example, in [25], the authors propose a fast and low complexity algorithm to deal with a combinatorial resource allocation model they propose, however they neither provide nor compare with the optimal solution of the problem. Some efforts have also been made while taking into account the stochastic nature of the problem (See [22,24]). The stochastic nature in the input data of an OFDMA model may arise from several sources. In particular, we consider the quality of the channels which directly affect the power consumption of the network and the bandwidth channel capacity required by each user [3]. In this paper, we propose a (0-1) two stage stochastic quadratic model for resource allocation in OFDMA networks. To this end, we suppose that only a subset of the input parameters are subject to uncertainty while the rest of them are considered to be deterministic. As far as we know, there are neither linear nor quadratic two stage stochastic formulations for resource allocation in OFDMA.

Stochastic programming (SP) is an optimization technique which deals with the uncertainty of the input parameters of a mathematical program [1]. These parameters can be modeled as random variables to which the theory of probabilities can be applied. This is an important difference when compared to robust optimization since the latter does not make use of explicit probability distributions [4]. The probability distributions governing the data are commonly known or can be estimated. It is frequently assumed that these probability distributions can take values within a discrete (finite) space. The latter allows considering finite sets of scenarios for each input parameter. There are two well known scenario based approaches in SP. The first one is known as *the recourse model* or *two stage stochastic* approach [4,14] while the second one is known as *probabilistic constrained* or *chance constrained* approach [2,21,14]. In the former, one must take a first stage decision without all the information in advance, i.e. on a here and now basis criteria. Although, the possibility to correct such decision in a second stage is possible. The probabilistic constrained approach, on the other hand, imposes a probability of occurrence for some (or all) of the constraints in a particular mathematical model. This means that some of the constraints will be satisfied for a given percentage while the rest of them will be ignored. The literature related to SP has grown considerably in last decades. A general survey can be found for instance in [21] and we refer the reader to [18,23,13] or to a more recent book [1] for a deeper comprehension. In this paper, we first assume that only the power behaves as a discrete random variable. And then, we study the more general case in which both, the power and the user bit rates behave as random variables. In the latter case, we introduce probabilistic constraints on each user bit rate which we handle by using the approach of [2]. The study is made on the basis of a binary quadratically constrained quadratic program (QCQP) we take from [15]. These formulations are all NP-Hard combinatorial optimization problems [3]. Therefore, we derive semidefinite programming relaxations (SDP) and linear programs (LP) to compute lower bounds. In particular, we derive the LP relaxations by using the Fortet linearization method [7]. Semidefinite programming

is a recently convex optimization approach concerned with the optimization of a linear cost function and linear constraints which are intersected with the cone of symmetric positive semidefinite matrices [10]. It has been shown that SDP can be highly effective when finding tight bounds for combinatorial optimization problems [16, 11, 9, 8]. Besides, SDP problems can be solved to optimality using polynomial interior point algorithms [11, 5]. The paper is organized as follows: section 2 briefly introduces the resource allocation problem and presents an OFDMA system description. In section 3, we present the two stage stochastic quadratic formulation with and without probabilistic constraints. Section 4 presents and explains the SDP and the LP relaxations we derive from the quadratic models. In section 5, we compute numerical results for both, the LP and SDP relaxations when compared to the optimal solution of the problem. Finally, section 6 gives the main conclusions of the paper.

2 Description of the problem

In this section, we briefly introduce the resource allocation problem in an OFDMA network and then, we provide a general system description of this type of network. As mention in section 1, the resource allocation problem has led to several mathematical formulations [3]. These formulations mainly differ in the objective function to be optimized, on the application to be performed, which can be either uplink (UL) or downlink (DL), and on the number of cells within the OFDMA network. In an UL application, the transmission of signals is performed from the users to the base station (BS) while in a DL application, this is done from the BS to the users. The number of cells, on the other hand, means that we can have an OFDMA system consisting of only one BS or the more general case composed of several BSs. The former is known as a single cell OFDMA network while the latter forms a multiple cell OFDMA network. The last one is by far the highest complex scenario since it involves, in addition, the interference generated among the different BSs. In this paper, we consider only the case of a single cell OFDMA network. Some examples of different mathematical formulations for OFDMA are: minimizing power subject to bit rates and sub-carrier allocation constraints, maximizing throughput subject to power constraints and minimizing BER¹ subject to power and quality of service (QoS) constraints [3]. Whatever approach is adopted, a common feature is that all these models are in general formulated as hard mixed integer nonlinear programming problems. The mathematical formulation we propose aims to minimize the total power consumption of a single cell OFDMA network subject to bit rates, sub-carrier and modulation constraints. The model can be applied to both types of applications, either UL or DL as it does not consider particular constraints on the users or in the BS.

A brief system description of the OFDMA network we study can be given as follows. Consider a BS surrounded by several mobile users within a single cell area. The BS has to assign a set $A = \{1, \dots, N\}$ of sub-carriers to a set of $B = \{1, \dots, K\}$ users while using a modulation size of $c \in C = \{1, \dots, M\}$ bits in each sub-carrier. In order to formulate a two stage stochastic model, we split the set A of sub-carriers into two subsets $A_1 = \{1, \dots, N_1\}$ and $A_2 = \{1, \dots, N_2\}$ such that $A = A_1 \cup A_2$. This could be possible if we assume that some of the linking channels going out from the subset A_2 are

¹ BER stands for *bit error rate* and measures the number of erroneous bits over all the bits sent through a particular telecommunication channel.

subject to uncertainty and thus harder to predict for the BS. Subset A_1 represents the sub-carriers which are not subject to uncertainty. In general, the power of an OFDMA network can be computed as [12]

$$P_{k,n}^c = \frac{\frac{N_0}{3} \left[Q^{-1} \left(\frac{BER_k}{4} \right) \right] (2^c - 1)}{\alpha_{k,n}} \quad (1)$$

where

$$Q(t) = \frac{1}{\sqrt{2\pi}} \int_t^\infty e^{-\frac{t^2}{2}} dt \quad (2)$$

This means that power is computed as a function of the modulation size of the bits $c \in C$ and of the quality of the linking channels $\alpha_{k,n}$ between user $k \in B$ and sub-carrier $n \in A$. In formula (1), BER_k represents the quality of service required by user k while N_0 denotes the power spectral density. Therefore, the power we use as input for our models is directly affected by the quality of the channel. The better the quality of the channel is, the less the amount of power we need to ensure that a signal will be detected with a given quality of service (BER) requirement.

3 Stochastic formulations

In this section, we propose a two stage stochastic quadratic program using subsets A_1 and A_2 as defined in section 2. Then, we further introduce a probabilistic constrained approach from [2] on the user bit rate constraints while using the same stochastic quadratic program.

3.1 A two stage stochastic formulation

The two stage stochastic model we propose can be formulated as follows

$$\text{TSQIP0: } \min \sum_{k=1}^K \sum_{n=1}^{N_1} \sum_{c=1}^M P_{k,n}^c x_{k,n} y_{n,c} + \mathbb{E}Q(\chi, x, y) \quad (3)$$

$$\text{st: } \sum_{k=1}^K x_{k,n} \leq 1 \quad \forall n \in A_1 \quad (4)$$

$$\sum_{c=1}^M y_{n,c} \leq 1 \quad \forall n \in A_1 \quad (5)$$

$$x \in \{0, 1\}^{KN_1}, y \in \{0, 1\}^{N_1M} \quad (6)$$

$$\text{where } Q(\chi, x, y) = \min \sum_{k=1}^K \sum_{n=1}^{N_2} \sum_{c=1}^M \tilde{P}_{k,n}^c(\chi) z_{k,n} w_{n,c} \quad (7)$$

$$\text{st: } \sum_{n=1}^{N_2} z_{k,n} \left[\sum_{c=1}^M c \cdot w_{n,c} \right] \geq R_k - \sum_{n=1}^{N_1} x_{k,n} \left[\sum_{c=1}^M c \cdot y_{n,c} \right], \quad \forall k \in B \quad (8)$$

$$\sum_{k=1}^K z_{k,n} \leq 1 \quad \forall n \in A_2 \quad (9)$$

$$\sum_{c=1}^M w_{n,c} \leq 1 \quad \forall n \in A_2 \quad (10)$$

$$z \in \{0, 1\}^{KN_2}, w \in \{0, 1\}^{N_2M} \quad (11)$$

In TSQIP0, (3)-(6) represent the first stage decision problem while (7)-(11), the second stage problem. In both problems, the objective functions (3) and (7) are intended to minimize the total power consumption of the OFDMA network. In TSQIP0, $(P_{k,n}^c)$ represents a deterministic power matrix while $(\tilde{P}_{k,n}^c(\chi))$ represents a stochastic power matrix. The latter depends on the random variable χ . The entries of the power matrices correspond to the amount of power we should assign to user k while using sub-carrier n and a modulation size of c bits. The term $\mathbb{E}Q(\chi, x, y)$ corresponds to the mathematical expectation of problem (7)-(11). Function $Q(\chi, x, y)$ also depends on the first stage decision variables $x_{k,n}$ and $y_{n,c}$. The decision variable $x_{k,n} = 1$ if user k is assigned sub-carrier n and $y_{n,c} = 1$ if sub-carrier n is assigned a modulation size of c bits. Constraints (4), (5) and (9), (10) represent sub-carrier and modulation linear constraints for the first and second stage problems, respectively. Finally, constraint (8) corresponds to the bit rate constraints of each user. This is a coupling constraint relating the first stage with the second stage decision variables $z_{k,n}$ and $w_{n,c}$. Similarly, $z_{k,n} = 1$ if user k is assigned sub-carrier n while $w_{n,c} = 1$ if sub-carrier n uses a modulation size of c bits, otherwise $z_{k,n} = 0$ or $w_{n,c} = 0$. In order to state an equivalent deterministic model for TSQIP0, we assume that the power matrix $\tilde{P}(\chi) = (\tilde{P}_{k,n}^c(\chi))$ is concentrated on a discrete finite number of scenarios as $\tilde{P}(\chi) = ([\tilde{P}_{k,n}^c]^1, \dots, [\tilde{P}_{k,n}^c]^s, \dots, [\tilde{P}_{k,n}^c]^S)$ with probabilities $\tilde{p}_s$, $s \in S$ such that $\sum_{s=1}^S \tilde{p}_s = 1$, $\tilde{p}_s \geq 0$, for all s . Putting it altogether, this leads us to write the following model

$$\begin{aligned} \text{TSQIP1: } \min \quad & \sum_{k=1}^K \sum_{n=1}^{N_1} \sum_{c=1}^M P_{k,n}^c x_{k,n} y_{n,c} + \\ & + \sum_{s=1}^S \tilde{p}_s \left(\sum_{k=1}^K \sum_{n=1}^{N_2} \sum_{c=1}^M [\tilde{P}_{k,n}^c]^s z_{k,n}^s w_{n,c}^s \right) \end{aligned} \quad (12)$$

$$\begin{aligned} \text{st: } \quad & \sum_{n=1}^{N_2} z_{k,n}^s \left[\sum_{c=1}^M c \cdot w_{n,c}^s \right] \geq R_k + \\ & - \sum_{n=1}^{N_1} x_{k,n} \left[\sum_{c=1}^M c \cdot y_{n,c} \right] \quad \forall k, s \end{aligned} \quad (13)$$

$$(4) - (6), \quad (14)$$

$$\sum_{k=1}^K z_{k,n}^s \leq 1 \quad \forall n \in A_2, s \in S \quad (15)$$

$$\sum_{c=1}^M w_{n,c}^s \leq 1 \quad \forall n \in A_2, s \in S \quad (16)$$

$$z \in \{0, 1\}^{KN_2S}, w \in \{0, 1\}^{N_2MS} \quad (17)$$

TSQIP1 is a deterministic equivalent quadratic problem which is NP-hard [15]. Therefore, we seek strong relaxations to find tight lower bounds, namely SDP relaxations. In section 5, we compare our SDP model with a Fortet LP equivalent model we obtain from TSQIP1 [7].

3.2 Probabilistic constraints on the user bit rates

In this subsection, we introduce the probabilistic constrained approach of [2] into TSQIP1. To this purpose, we transform each constraint in (13) from TSQIP1 into a probabilistic constraint such as

$$\mathbb{P} \left\{ \sum_{n=1}^{N_2} z_{k,n}^s \left[\sum_{c=1}^M c \cdot w_{n,c}^s \right] \geq R_k(\omega) - \sum_{n=1}^{N_1} x_{k,n} \left[\sum_{c=1}^M c \cdot y_{n,c} \right] \right\} \geq (1 - \beta_k),$$

$$\forall k \in B, s \in S \quad (18)$$

This can be done if we assume that the bit rate of each user behaves as a random variable, i.e. $R_k = R_k(\omega)$. Similarly, as we do for the power in the above subsection, we may suppose that the random variable $R_k(\omega)$ is distributed according to a discrete probability distribution. This allows us to consider the finite set of scenarios $R_k(\omega) = (R_{k,1}, \dots, R_{k,L_k})$ with probabilities $p_{k,l}$ such that $\sum_{l=1}^{L_k} p_{k,l} = 1, p_{k,l} \geq 0$, for all $k \in B$. Using this set of scenarios, we may rewrite model TSQIP0 according to [2] as follows

$$\begin{aligned} \min \quad & (12) \\ \text{st:} \quad & (4) - (6), (15) - (17) \\ & \sum_{n=1}^{N_2} z_{k,n}^s \left[\sum_{c=1}^M c \cdot w_{n,c}^s \right] \geq R_{k,l} - \sum_{n=1}^{N_1} x_{k,n} \left[\sum_{c=1}^M c \cdot y_{n,c} \right], \\ & \forall k \in B, s \in S, l \in \Gamma_k \end{aligned} \quad (19)$$

$$\sum_{l \in \Gamma_k} p_{k,l} \geq (1 - \beta_k), \forall k \in B \quad (20)$$

Constraints in (20) mean that we have to choose a subset Γ_k of scenarios such that the sum of the probabilities of this subset is greater than a given percentage $(1 - \beta_k)$ where $\beta_k \in (0, 0.5]$. For this subset, the bit rate constraints will be included, whereas for the scenarios not in this subset, the constraints are not activated. This problem can be reformulated by introducing the auxiliary binary variable $\varphi_{k,l}$ for each observation $l = 1 : L_k$ and for each k as follows:

$$\varphi_{k,l} = \begin{cases} 0 & \text{if } l \in \Gamma_k \\ 1 & \text{otherwise} \end{cases}$$

This yields the following problem

$$\begin{aligned} \text{TSQIP2:} \quad \min \quad & (12) \\ \text{st:} \quad & (4) - (6), (15) - (17) \\ & \sum_{n=1}^{N_2} z_{k,n}^s \left[\sum_{c=1}^M c \cdot w_{n,c}^s \right] \geq R_{k,l} - \sum_{n=1}^{N_1} x_{k,n} \left[\sum_{c=1}^M c \cdot y_{n,c} \right] - \mathcal{M} \varphi_{k,l}, \\ & \forall k \in B, s \in S, l \in \{1, \dots, L_k\} \end{aligned} \quad (21)$$

$$\sum_{l=1}^{L_k} p_{k,l} \cdot \varphi_{k,l} \leq \beta_k, \quad \forall k \in B \quad (22)$$

$$\varphi_{k,l} \in \{0, 1\}^{KL_k} \quad (23)$$

where $\mathcal{M}$ is an arbitrary number such that

$$\mathcal{M} \geq \max_{k,l} \{r_{k,l}\}$$

The parameter $\mathcal{M}$ will be active only if variable $\varphi_{k,l} = 1$ which means we force a particular constraint not to be satisfied.

In the next section, we propose SDP and LP relaxations to compute lower bounds for the quadratic models we formulate in this section. Computing tight lower and upper bounds constitute a major step towards finding the optimal solution of a combinatorial optimization problem. In particular, computing tight feasible solutions would serve to prune non-valuable subtrees in a branch-and-bound framework [19]. In this paper, we mainly focus on deriving tight SDP lower bounds as they could be used to find tight upper bounds as well. For instance, in [15] the authors obtain tight feasible solutions and sometimes the optimal solution of the problem by simply using a randomized rounding procedure.

4 Semidefinite and linear programming relaxations

We first propose two SDP relaxations, one for TSQIP1 and another one for TSQIP2, respectively. Then, we derive for each quadratic model (TSQIP1 and TSQIP2), two LP relaxations using Fortet linearization method [7].

4.1 Semidefinite programming relaxations

To state a SDP relaxation for TSQIP1, we define the (0-1) vectors

$$d^T = (x_{1,1}, \dots, x_{1,N_1}, \dots, x_{K,1}, \dots, x_{K,N_1}, y_{1,1}, \dots, y_{1,M}, \dots, \\ \dots, y_{N_1,1}, \dots, y_{N_1,M})$$

$$e_s^T = (z_{1,1}^s, \dots, z_{1,N_2}^s, \dots, z_{K,1}^s, \dots, z_{K,N_2}^s, w_{1,1}^s, \dots, w_{1,M}^s, \dots, \\ \dots, w_{N_2,1}^s, \dots, w_{N_2,M}^s), \quad \forall s \in S$$

and the extended binary vector

$$f^T = (d, e_1, e_2, \dots, e_S) \quad (24)$$

Now, let X be a symmetric positive semidefinite matrix defined as

$$X = \begin{pmatrix} f f^T & f \\ f^T & 1 \end{pmatrix} \succeq 0$$

We may construct input symmetric matrices $\mathcal{P}$ for the objective function in TSQIP1 and $U_{k,s}$ for constraints in (13). Note that constraints (4),(5),(15) and (16) are all linear

and then we can use lemma 2.1 from [10] to strengthen them. Thus, we define the coefficient vectors $\mu_n, \nu_n, \lambda_{n,s}, \omega_{n,s}$ according to vector f for each constraint (4),(5),(15) and (16), respectively. This allows writing them as $\mu_n^T f \leq 1, \nu_n^T f \leq 1, \lambda_{n,s}^T f \leq 1$ and $\omega_{n,s}^T f \leq 1$, respectively. Whereby squaring both sides of each inequality, we may obtain $Trace([\mu_n][\mu_n]^T f f^T) \leq 1, Trace([\nu_n][\nu_n]^T f f^T) \leq 1, Trace([\lambda_{n,s}][\lambda_{n,s}]^T f f^T) \leq 1$ and $Trace([\omega_{n,s}][\omega_{n,s}]^T f f^T) \leq 1$. Here, the $Trace(\cdot)$ operator represents the Frobenius inner product between two symmetric matrices. Finally, let $[\mu_n][\mu_n]^T, [\nu_n][\nu_n]^T, [\lambda_{n,s}][\lambda_{n,s}]^T$ and $[\omega_{n,s}][\omega_{n,s}]^T$ be the matrices we construct while adding as many as necessary zero row-column vectors to matrices $[\mu_n][\mu_n]^T, [\nu_n][\nu_n]^T, [\lambda_{n,s}][\lambda_{n,s}]^T$ and $[\omega_{n,s}][\omega_{n,s}]^T$ until reaching the appropriate size of X . Hence, we come up with the following SDP relaxation for TSQIP1

$$\begin{aligned}
 \text{TSSDP1: } &= \min \quad Trace(\mathcal{P}X) \\
 \text{st: } &Trace(U_{k,s}X) \geq r_k \quad \forall k \in B, s \in S \\
 &Trace([\mu_n][\mu_n]^T X) \leq 1 \quad \forall n \in A_1 \quad (25) \\
 &Trace([\nu_n][\nu_n]^T X) \leq 1 \quad \forall n \in A_1 \quad (26) \\
 &Trace([\lambda_{n,s}][\lambda_{n,s}]^T X) \leq 1 \quad \forall n \in A_2, s \in S \quad (27) \\
 &Trace([\omega_{n,s}][\omega_{n,s}]^T X) \leq 1 \quad \forall n \in A_2, s \in S \quad (28) \\
 &Trace(\Pi_{k,n}^c X) \geq 0 \quad \forall k \in B, c \in C, n \in A_1 \quad (29) \\
 &Trace([\zeta_{k,n}^c]^s X) \geq 0, \\
 &\forall k \in B, c \in C, n \in A_2, s \in S \quad (30) \\
 &diag(f f^T) = f \quad (31) \\
 &X \succeq 0 \quad (32)
 \end{aligned}$$

In TSSDP1, constraints (31) and (32) form a relaxation constraint for the condition of $f_i \in \{0, 1\}$ for all i . Finally, the input symmetric matrices $\Pi_{k,n}^c$ and $[\zeta_{k,n}^c]^s$ are used to have positive values in matrix X only in the positions where $\{\mathcal{P}_{i,j}, i < j\}$ is positive.

To obtain a SDP relaxation for TSQIP2, we define the binary vectors g_k for all $k \in B$ as

$$g_k^T = (\varphi_{k,1}, \dots, \varphi_{k,L_k}) \quad \forall k \in B$$

Then, we use vector f as defined in (24) to construct a new larger (0-1) vector h as follows

$$h^T = (f, g_1, \dots, g_K) \quad (33)$$

Using vector h , we also define a new larger positive semidefinite variable matrix $\tilde{X}$ as

$$\tilde{X} = \begin{pmatrix} h h^T & h \\ h^T & 1 \end{pmatrix} \succeq 0$$

and then, analogously to the way we have constructed TSSDP1, we may define input symmetric matrices $\tilde{\mathcal{P}}$ for the objective function in TSQIP2 and $\tilde{U}_{k,s}^l$ for each constraint in (21), but now according to matrix $\tilde{X}$. Similarly, we denote by $[\mu_n][\mu_n]^T, [\nu_n][\nu_n]^T, [\lambda_{n,s}][\lambda_{n,s}]^T$ and $[\omega_{n,s}][\omega_{n,s}]^T$ the input symmetric matrices we use for constraints

(4),(5),(15) and (16) in TSQIP2. These matrices are extended matrices we form from $[\mu_n][\mu_n]^T$, $[\nu_n][\nu_n]^T$, $[\lambda_{n,s}][\lambda_{n,s}]^T$ and $[\omega_{n,s}][\omega_{n,s}]^T$ while adding as many as zero row-column vectors to get the appropriate size of matrix $\tilde{X}$. Additionally, we introduce a new vector δ_k for each probabilistic constraint in (22) and use lemma 2.1 from [10] to strengthen these probabilistic constraints as well. This yields the following SDP relaxation for TSQIP2

$$\text{TSSDP2:} = \min \text{Trace}(\tilde{\mathcal{P}}\tilde{X})$$

$$\text{st: } \text{Trace}(\tilde{U}_{k,s}^l \tilde{X}) \geq \tilde{r}_{k,l} \quad \forall k \in B, s \in S, l = 1 : L_k \quad (34)$$

$$\text{Trace}([\delta_k][\delta_k]^T \tilde{X}) \leq \beta_k^2 \quad \forall k \in B \quad (35)$$

$$\text{Trace}(\widetilde{[\mu_n][\mu_n]^T} \tilde{X}) \leq 1 \quad \forall n \in A_1 \quad (36)$$

$$\text{Trace}(\widetilde{[\nu_n][\nu_n]^T} \tilde{X}) \leq 1 \quad \forall n \in A_1 \quad (37)$$

$$\text{Trace}(\widetilde{[\lambda_{n,s}][\lambda_{n,s}]^T} \tilde{X}) \leq 1 \quad \forall n \in A_2, s \in S \quad (38)$$

$$\text{Trace}(\widetilde{[\omega_{n,s}][\omega_{n,s}]^T} \tilde{X}) \leq 1 \quad \forall n \in A_2, s \in S \quad (39)$$

$$\text{Trace}(\tilde{H}_{k,n}^c \tilde{X}) \geq 0 \quad \forall k \in B, c \in C, n \in A_1 \quad (40)$$

$$\text{Trace}([\tilde{\zeta}_{k,n}^c]^s \tilde{X}) \geq 0, \quad (41)$$

$$\forall k \in B, c \in C, n \in A_2, s \in S$$

$$\text{diag}(h h^T) = h \quad (42)$$

$$\tilde{X} \succeq 0 \quad (43)$$

Similarly as done for TSSDP1, the input matrices $\tilde{H}_{k,n}^c$ and $[\tilde{\zeta}_{k,n}^c]^s$ within constraints (40) and (41) force the fact that $\tilde{X}$ must have positive values in the entries where $\{\tilde{\mathcal{P}}_{i,j}, i < j\}$ is positive. Note, we do not use these type of constraints for variables $\varphi_{k,l}$. It turns out that only the diagonal and last column within this part of matrix $\tilde{X}$ containing variables $\varphi_{k,l}$ are used. Besides, constraints (42) and (43) state the condition of $\tilde{X}_{i,i} = \tilde{X}_{i,F}$ for all i where F represents the index of the last column. The latter is also valid for variables $\varphi_{k,l}$ as well.

4.2 Linear programming relaxations

In this subsection, we derive a LP relaxation for each quadratic model TSQIP1 and TSQIP2, respectively. These LP models are obtained using Fortet linearization method [7]. The simplest and provably the easiest way to derive an equivalent integer linear program (IP) from a QCQP is the classic Fortet linearization method. This method mainly consists of replacing each product term of a QCQP by an additional variable while simultaneously adding some linear constraints. To apply this method, we introduce linearization variables $\psi_{k,n}^c = x_{k,n} y_{n,c}$ and $\vartheta_{k,n}^{c,s} = z_{k,n}^s w_{n,c}^s$. This yields the following model we call hereafter TSIP1_F

$$\text{TSIP1}_F: \min \sum_{k=1}^K \sum_{n=1}^{N_1} \sum_{c=1}^M P_{k,n}^c \psi_{k,n}^c +$$

$$+ \sum_{s=1}^S \tilde{p}_s \left(\sum_{k=1}^K \sum_{n=1}^{N_2} \sum_{c=1}^M [\tilde{P}_{k,n}^c]^s \vartheta_{k,n}^{c,s} \right) \quad (44)$$

$$\begin{aligned}
 \text{st: } & \sum_{n=1}^{N_2} \left[\sum_{c=1}^M c \cdot \vartheta_{k,n}^{c,s} \right] \geq R_k - \\
 & \quad + \sum_{n=1}^{N_1} \left[\sum_{c=1}^M c \cdot \psi_{k,n}^c \right] \quad \forall k, s \quad (45) \\
 & (4) - (6), (15) - (17) \\
 & \psi_{k,n}^c \leq x_{k,n}, \psi_{k,n}^c \leq y_{n,c}, \\
 & \psi_{k,n}^c \geq x_{k,n} + y_{n,c} - 1, \quad \forall k, n, c \quad (46) \\
 & \vartheta_{k,n}^{c,s} \leq z_{k,n}^s, \vartheta_{k,n}^{c,s} \leq w_{n,c}^s, \\
 & \vartheta_{k,n}^{c,s} \geq z_{k,n}^s + w_{n,c}^s - 1, \quad \forall k, n, c, s \quad (47) \\
 & \psi_{k,n}^c \in \{0, 1\}^{KN_1M}, \vartheta_{k,n}^{c,s} \in \{0, 1\}^{KN_2MS} \quad (48)
 \end{aligned}$$

In TSIP1_F, constraints (46)-(47) are the Fortet linearization constraints. Hereafter, we denote by TSLP1_F the LP relaxation we obtain from TSIP1_F by relaxing its integer variables in the interval [0, 1].

To derive a Fortet IP model for TSQIP2, we use the same linearization variables $\psi_{k,n}^c$ and $\vartheta_{k,n}^{c,s}$. Thus, a Fortet IP model for TSQIP2 can be written as follows

$$\begin{aligned}
 \text{TSIP2}_F: \quad & \min \quad (44) \quad (49) \\
 \text{st: } & \sum_{n=1}^{N_2} \left[\sum_{c=1}^M c \cdot \vartheta_{k,n}^{c,s} \right] \geq R_{k,l} - \sum_{n=1}^{N_1} \left[\sum_{c=1}^M c \cdot \psi_{k,n}^c \right] - \mathcal{M}\varphi_{k,l} \\
 & \forall k \in B, s \in S, l \in \{1, \dots, L_k\} \quad (50) \\
 & (4) - (6), (15) - (17), (22) - (23), (46) - (48)
 \end{aligned}$$

Similarly, we denote by TSLP2_F the LP relaxation obtained from TSIP2_F while bounding its variables inside the interval [0, 1].

5 Numerical results

In this section, we present numerical results for the integer linear programs TSIP1_F, TSIP2_F, for their LP relaxations TSLP1_F, TSLP2_F and for the SDP relaxations TSSDP1 and TSSDP2, respectively. We generate power data using the power formula (1) for wireless channels [12]. A Matlab program is developed using CPLEX 12 and CSDP 6.01 [5] to solve the IP, LP and SDP relaxations. The numerical experiments have been carried out on a Pentium IV, 1 GHz with 2 GoBytes of RAM under windows XP. The size of the instances we run for the LP and SDP relaxations are shown in table 1. The first column gives the instance number while columns 2 to 6 give the size of each instance. Columns 7-8 and 11-12 give the number of constraints and variables for the LP relaxations TSLP1_F and TSLP2_F. The same information is given for TSSDP1 and TSSDP2 in columns 9-10 and 13-14, respectively. From this table, we observe that the number of constraints grows faster for the LP relaxation than for the SDP relaxations. This is mainly due to the fact that we add Fortet linear constraints. On the opposite, the number of variables is larger for the SDP relaxations. This can be explained by the order size of the variable matrix within the SDP relaxations which is increased in one unit per variable. We simulate our instances varying the number of users for different

Size of the Instances						TSLP1 _F		TSSDP1		TSLP2 _F		TSSDP2	
#	K	N1	N2	S	L	# Const.	# Var.	# Const.	# Var.	# Const.	# Var.	# Const.	# Var.
1	4	8	8	2	2	2360	576	633	18721	2388	584	653	20301
2	6	8	8	2	2	3420	816	877	29161	3462	828	907	32131
3	8	8	8	2	2	4480	1056	1121	41905	4536	1072	1161	46665
4	10	8	8	2	2	5540	1296	1365	56953	5610	1316	1415	63903
5	12	8	8	2	2	6600	1536	1609	74305	6684	1560	1669	83845
6	4	8	8	4	4	3936	960	1057	51681	4020	976	1125	56953
7	6	8	8	4	4	5704	1360	1465	80601	5830	1384	1567	90525
8	8	8	8	4	4	7472	1760	1873	115921	7640	1792	2009	131841
9	10	8	8	4	4	9240	2160	2281	157641	9450	2200	2451	180901
10	12	8	8	4	4	11008	2560	2689	205761	11260	2608	2893	237705
11	4	16	16	2	2	4712	1152	1257	74305	4740	1160	1277	77421
12	6	16	16	2	2	6828	1632	1741	115921	6870	1644	1771	121771
13	8	16	16	2	2	8944	2112	2225	166753	9000	2128	2265	176121
14	10	16	16	2	2	11060	2592	2709	226801	11130	2612	2759	240471
15	12	16	16	2	2	13176	3072	3193	296065	13260	3096	3253	314821
16	4	16	16	4	4	7856	1920	2097	205761	7940	1936	2165	216153
17	6	16	16	4	4	11384	2720	2905	321201	11510	2744	3007	340725
18	8	16	16	4	4	14912	3520	3713	462241	15080	3552	3849	493521
19	10	16	16	4	4	18440	4320	4521	628881	18650	4360	4691	674541
20	12	16	16	4	4	21968	5120	5329	821121	22220	5168	5533	883785
21	4	32	32	2	2	9416	2304	2505	296065	9444	2312	2525	302253
22	6	32	32	2	2	13644	3264	3469	462241	13686	3276	3499	473851
23	8	32	32	2	2	17872	4224	4433	665281	17928	4240	4473	683865
24	10	32	32	2	2	22100	5184	5397	905185	22170	5204	5447	932295
25	12	32	32	2	2	26328	6144	6361	1181953	26412	6168	6421	1219141
26	4	32	32	4	4	15696	3840	4177	821121	15780	3856	4245	841753
27	6	32	32	4	4	22744	5440	5785	1282401	22870	5464	5887	1321125
28	8	32	32	4	4	29792	7040	7393	1846081	29960	7072	7529	1908081
29	10	32	32	4	4	36840	8640	9001	2512161	37050	8680	9171	2602621
30	12	32	32	4	4	43888	10240	10609	3280641	44140	10288	10813	3404745

Table 1 Instance Sizes for the LP and SDP Relaxations

fixed number of sub-carriers N_1 and N_2 , respectively. This is a valid assumption in OFDMA networks since the bandwidth of a single channel can span from 1.25MHz to 20MHz. And this is directly linked to the number of sub-carriers which commonly take values of $N_1, N_2 = \{8, 16, 32, 64, \dots\}$ [3]. The maximum number of bits to be transmitted in each sub-carrier is set to $M = 4$. These are also common modulation sizes when using M-PSK or M-QAM modulations in OFDMA systems [6].

The numerical results we obtain when solving the instances of table 1 are shown in tables 2 and 3, respectively. Both tables provide the same column information. Column 1 gives the instance number according to table 1. Column 2 gives the optimal solution of the equivalent IP Fortet model TSIP1_F (Resp. TSIP2_F). In columns 3 and 4, we provide the lower bounds and Cpu Times in seconds for TSLP1_F (Resp. TSLP2_F). Columns 5 and 6 provide lower bounds and Cpu Times (in seconds) for TSSDP1 (Resp. TSSDP2). Finally, integrality gaps we compute as $\left(\frac{TSIP1 - TSLP1}{TSIP1}\right) \cdot 100$ and $\left(\frac{TSIP1 - TSSDP1}{TSIP1}\right) \cdot 100$ (Resp. using TSIP2, TSLP2 and TSSDP2) for the LP and SDP relaxations are given by columns 7 and 8. From table 2 we can see that the SDP bounds are tighter than the LP ones for all the instances. Furthermore, TSSDP1 gives near optimal bounds when compared to the objective value function of the problem. This can be better shown by computing the average gaps among the 30 instances of table 2 which are 2.07 % for TSSDP1 and 16.02 % for the TSLP1_F. Within this 2.07 %, we have zero gaps for instances 6, 11, 21 and 26. In general, we obtain better lower bounds for both the LP and SDP relaxations when the ratio $\left(\frac{N_1 + N_2}{K}\right)$ is larger. This is an interesting observation since the relation between the number of sub-carriers and users in OFDMA networks usually satisfy $N \gg K$ [27,3]. Regarding the cpu times, we have the opposite situation. Clearly, the SDP relaxations are harder to solve than the LP ones. The average Cpu time for the TSSDP1 is 2828.7 while for TSLP1_F is 1.59 seconds. Until now, this is the main drawback when solving SDP relaxations. Although, interior point algorithms have polynomial complexity, they still can not solve

#	LP Relaxation			SDP Relaxation		Gaps (%)	
	TSLP1 _F	TSLP1 _F	Cpu Time (s)	TSSDP1	Cpu Time (s)	LP	SDP
1	0.3134	0.2939	1.01	0.3123	13.15	6.21	0.36
2	0.1428	0.1284	1.03	0.1389	33.65	10.10	2.70
3	0.2743	0.1916	1.06	0.2601	37.65	30.16	5.18
4	0.0922	0.0694	1.14	0.0861	75.65	24.75	6.57
5	0.1543	0.1109	1.03	0.1477	119.92	28.12	4.28
6	0.1851	0.1224	0.96	0.1851	37.34	33.88	0
7	0.0725	0.0620	1.32	0.0680	85.53	14.49	6.11
8	0.4247	0.2726	1.28	0.3964	171.70	35.80	6.66
9	0.0992	0.0835	1.32	0.0984	309.06	15.77	0.74
10	0.2271	0.1742	1.45	0.2081	504.03	23.29	8.33
11	0.0819	0.0774	1.03	0.0819	80.53	5.44	0
12	0.1686	0.1300	1.10	0.1631	216.54	22.93	3.27
13	0.0774	0.0694	1.26	0.0772	272.18	10.29	0.21
14	0.1295	0.1119	1.35	0.1291	489.56	13.55	0.33
15	0.1047	0.0913	1.95	0.1042	797.57	12.75	0.53
16	0.1916	0.1811	1.48	0.1913	341.75	5.46	0.13
17	0.1908	0.1452	1.42	0.1904	572.17	23.90	0.20
18	0.1122	0.0978	1.39	0.1027	1294.1	12.79	8.46
19	0.2250	0.1851	1.76	0.2234	2198.6	17.69	0.70
20	0.0634	0.0599	3.26	0.0632	5200	5.45	0.36
21	0.1681	0.1604	2.20	0.1681	1629	4.57	0
22	0.1282	0.1149	1.42	0.1276	1643.5	10.36	0.47
23	0.1092	0.1017	1.51	0.1092	1911.9	6.85	0
24	0.1714	0.1407	1.64	0.1708	3541.8	17.93	0.39
25	0.0788	0.0715	1.68	0.0785	5548.6	9.26	0.41
26	0.0964	0.0755	1.31	0.0964	1395.4	21.67	0
27	0.1245	0.1191	1.50	0.1226	3434.5	4.33	1.55
28	0.2519	0.2001	3.17	0.2451	7300	20.53	2.68
29	0.0947	0.0870	2.48	0.0943	13275	8.16	0.46
30	0.1440	0.1092	3.39	0.1422	32330	24.18	1.28

Table 2 Lower bounds for TSLP1_F and TSSDP1

#	LP Relaxation			SDP Relaxation		Gaps (%)	
	TSLP2 _F	TSLP2 _F	Cpu Time (s)	TSSDP2	Cpu Time (s)	LP	SDP
1	0.4567	0.2056	1.56	0.2558	44.04	54.98	43.98
2	0.1328	0.0883	1.53	0.1123	112.43	33.48	15.42
3	0.1535	0.1101	1.62	0.1420	101.15	28.24	7.44
4	0.1434	0.1063	1.68	0.1336	176.07	25.86	6.85
5	0.2908	0.1423	2.01	0.2001	293.90	51.05	31.19
6	0.1984	0.0977	1.87	0.1235	127.25	50.73	37.74
7	0.2223	0.1411	1.76	0.1676	295.31	36.53	24.62
8	0.3957	0.1955	1.96	0.2557	643.65	50.59	35.38
9	0.1743	0.1046	2.07	0.1276	994.67	39.99	26.78
10	0.2909	0.1497	2.96	0.2130	1577.5	48.54	26.78
11	0.2537	0.1622	1.87	0.2086	271.59	36.06	17.78
12	0.0997	0.0836	1.70	0.0928	781.71	16.17	6.97
13	0.1181	0.0710	2.03	0.0985	671.40	39.88	16.59
14	0.1296	0.0883	2.23	0.1006	983.59	31.87	22.41
15	0.1912	0.1119	2.45	0.1319	1632.6	41.46	31.00
16	0.1976	0.0901	2.29	0.1209	731.15	54.39	38.80
17	0.2632	0.1446	2.09	0.1860	1950.6	45.04	29.32
18	0.1983	0.1017	2.87	0.1334	3663.2	48.70	32.72
19	0.2443	0.1234	3.34	0.1773	6245.5	49.49	27.44
20	0.2895	0.1589	2.60	0.2063	4985.2	45.10	28.73
21	0.1992	0.1562	1.23	0.1754	958.89	21.59	11.92
22	0.1077	0.0800	1.32	0.0947	2643.1	25.71	12.05
23	0.2321	0.1536	1.68	0.1943	2394.5	33.84	16.30
24	0.1366	0.0982	2.18	0.1197	4572.5	28.11	12.37
25	0.2403	0.1565	2.11	0.2066	7453.3	34.86	14.00
26	0.1282	0.0742	1.54	0.0896	2904.9	42.15	30.14
27	0.2082	0.1256	2.68	0.1510	5939.1	39.66	27.45
28	0.1936	0.1075	2.25	0.1357	12735	44.46	29.92
29	0.2638	0.1143	5.26	0.1431	34720	56.69	45.76
30	0.3518	0.1617	7.62	0.2086	55370	54.03	40.69

Table 3 Lower bounds for TSLP2_F and TSSDP2

SDP relaxations as fast as we wish. However, we believe that in the upcoming years, fast and low complexity algorithms will be available to solve SDP relaxations (see for instance [26,17]). Besides, as we point out in section 1, in this paper, we are mainly concerned with solving the problem from a combinatorial optimization point of view rather than from a technological perspective.

Regarding table 3, we observe similar trends. In general, SDP lower bounds are tighter than the LP ones, although they are not near optimal now. In this case, we compute an average gap of 40.30 % for TSLP2_F and 24.95 % for TSSDP2. Again, we observe that while the ratio $\left(\frac{N_1+N_2}{K}\right)$ is larger, the better the bounds we get. Concerning the cpu times, we have the same trends as in table 2, i.e. CSDP needs a higher computational time to compute a solution than CPLEX needs to solve the LP relaxation. In this case, the average gives 5199.1 for TSSDP2 and 2.34 seconds for TSLP2_F. The increment of TSSDP2 over TSSDP1 can be explained by the larger size of the variable matrix $\tilde{X}$ of TSSDP2 compared to matrix X in TSSDP1 which causes CSDP solver to run slower [5]. Finally, we observe from tables 2 and 3 that the effect of increasing the number of scenarios in the second stage problem worsen both, the LP and SDP bounds.

6 Conclusions

In this paper, we proposed a two stage stochastic quadratic model for wireless OFDMA networks. The main goal is to minimize the total power consumption of the network subject to bit rate, sub-carrier and modulation constraints. The quadratic model is formulated as a two stage stochastic program in which only the input power is assumed to be affected by uncertainty. Afterward, we study the more general case in which both, the power and the bit rates of each user are considered to be discrete random variables. Thus, we impose on each user bit rate constraint a given probability of occurrence. The probabilistic constraints are transformed into deterministic constraints by using the approach of [2]. To compute lower bounds, we derive SDP and LP relaxations for these quadratic models. Our numerical results showed tight bounds for the SDP relaxation in contrast to those obtained with the LP relaxations. Although at a higher computational cost. The best results we obtain, which are near optimal, are those obtained using the SDP relaxation without probabilistic constraints.

Acknowledgements The authors are grateful for the financial support given by Conicyt Chilean government through the Insertion project number: 79100020 and by Usach university through DICYT project number 061117S.

References

1. Alexander Shapiro, Darinka Dentcheva and Andrzej Ruszczyński, Lectures on Stochastic Programming: Modeling and Theory, 436. SIAM Philadelphia, Series on Optimization, **9** of MPS/SIAM, Philadelphia (2009).
2. Alexei Gaivoronski, Abdel Lisser, Rafael Lopez and Hu Xu, Knapsack problem with probability constraints, Journal of Global Optimization, **49**, 397-413 (2011).
3. Amzallag, D. Armarnik, T. Livschitz and M. Raz, D., Multi-Cell Slots Allocation in OFDMA Systems, In Mobile and Wireless Communications Summit, 16th IST (2007).
4. Bertsimas Dimitris and Vineet Goyal, On the Power of Robust Solutions in Two-Stage Stochastic and Adaptive Optimization Problems, Mathematics of Operations Research, **35**, 284-305 (2010).
5. B. Borchers, CSDP, A C Library for Semidefinite Programming, Optimization Methods and Software **11**, 613-623 (1999).
6. Chadi Tarhini and Tijani Chahed, On capacity of OFDMA-based IEEE802.16 WiMAX including Adaptive Modulation and Coding (AMC) and inter-cell interference, In 15th IEEE Workshop on Local and Metropolitan Area Networks, 139-144 (2007).

7. Fortet R., Applications de l'algebre de boole en recherche operationelle, *Revue Francaise de Recherche Operationelle*, **4**, 17-26 (1960).
8. Goemans, M. X. and Williamson, D.P., Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *J. ACM.*, 1115-1145 (1995).
9. Grötschel, Lovász and Schrijver, *Geometric Algorithms and Combinatorial Optimization*, 362. Springer-Verlag, Berlin Heidelberg, Germany (1988).
10. Helmberg, C., F. Rendl and R. Weismantel, A Semidefinite Programming Approach to the Quadratic Knapsack Problem, *Journal of combinatorial optimization*, **4**, 197-215 (2000).
11. Helmberg, C., Semidefinite Programming for Combinatorial Optimization, 150. ZIB-Report ZR-00-34, Konrad-Zuse-Zentrum Berlin, Berlin Germany, October (2000).
12. I. Kim, H. L. Lee, B. Kim and Y. H. Lee, On the use of linear programming for dynamic subchannel and bit allocation in multiuser OFDM, In *IEEE GLOBECOM 01*, **6**, 3648-3652 (2001).
13. John Birge and Francois Louveaux, *Introduction to stochastic programming*. Springer-Verlag, New York (1997).
14. Lisser A., Lopez R. and Hu Xu, Stochastic Quadratic Knapsack with Recourse, In *International Network Optimization Conference (INOC-2009)*, April (2009).
15. Pablo Adasme, Abdel Lisser and Ismael Soto, Robust semidefinite relaxations for a quadratic OFDMA resource allocation scheme, *Computers & Operations Research*, **38**, 1377-1399 (2011).
16. Pablo Adasme and Abdel Lisser, Semidefinite and Conic Programming for Robust wireless OFDMA networks, *Electronic Notes in Discrete Mathematics*, **36**, 1225-1232 (2010).
17. Sanjeev Arora, Elad Hazan and Satyen Kale, Fast algorithms for approximate semidefinite programming using the multiplicative weights update method, In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, 339-348 (2005).
18. Schultz Rüdiger, Stougie Leen and Maarten H. Van Der Vlerk, Two-stage stochastic integer programming: a survey, *Statistica Neerlandica*, **50**, 404-416 (1996).
19. S. Kosuch and A. Lisser, On two-stage stochastic knapsack problems, To appear in *Discrete Applied Mathematics* (2010).
20. T. Keller and L. Hanzo., Adaptive multicarrier modulation: A convenient framework for time frequency processing in wireless communications, In *Proceedings IEEE*, **88**, 611-640 (2000).
21. V. Nikolaos Sahinidis, Optimization under uncertainty: State-of-the-art and opportunities, *Computers and Chemical Engineering*, **28**, 971-983 (2004).
22. Wei Liang Li, Ying Jun Zhang and Win, M.Z., Slow Adaptive OFDMA via Stochastic Programming, In *Communications ICC '09, IEEE International Conference on*, 1-6 (2009).
23. Willem K. Klein Haneveld and Maarten H. van der Vlerk, Stochastic integer programming: general models and algorithms, *Ann. Oper. Res.*, **85**, 39-57 (1999).
24. Wong, I.C. and Evans, B.L., Optimal resource allocation in the OFDMA downlink with imperfect channel knowledge, *IEEE Transactions on Communications*, **57** (2009).
25. Wong, I.C., Zukang Shen, Evans, B.L. and Andrews, J.G., A low complexity algorithm for proportional resource allocation in OFDMA systems, In *IEEE Workshop on Signal Processing Systems*, 1-6 (2004).
26. Zaiwen Wen, Donald Goldfarb, and Wotao Yin, Alternating Direction Augmented Lagrangian Methods for Semidefinite Programming, submitted to *Mathematical Programming Computation* (2010).
27. 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).

A Multi Species Evolutionary Algorithm for Tool Group Scheduling in Semiconductor Manufacturing

Tobias Uhlig · Oliver Rose

Abstract In semiconductor manufacturing, we face many complex scheduling problems, which are difficult to tackle with conventional optimization techniques. Evolutionary algorithms are a reasonable method to approach these problems. However, their quality depends on the adequateness of the employed problem encoding and determining the best problem representation is a big challenge. Therefore we propose to use a number of different representations (species), competing with each other in an evolutionary algorithm. We developed an extensible evolutionary algorithm, which adheres to a modular design philosophy enabling customization in a multitude of ways. It supports multiple species and races, different selection strategies and further extensions like aging. A typical problem we investigate is offline scheduling for parallel cluster tools, which effectively is a combination of partitioning and sequencing of jobs with additional constraints. In this paper we try to maximize the throughput of a cluster tool group by minimizing the makespan of the scheduled jobs. The multi-species approach we employed generated promising results for this particular problem.

1 Introduction to Evolutionary Algorithms

Evolutionary Algorithms (EAs) are widely used to generate solutions to optimization and search problems, since they return good results for a large range of problems [1, p. 17]. For this purpose they rely on repeated application of two fundamental concepts - variation and selection. Variation generates new solutions derived from existing ones. It increases the diversity in a population and facilitates the exploration of the search space. The variation operators of EAs are mutation and recombination. Selection is the driving force of evolution. Since variation by itself is not goal driven, selection is employed to increase the average quality of populations in the course of generations. This is achieved by favoring fitter individuals, which represent better solutions to a given problem. Typically selection methods

Tobias Uhlig
Dresden University of Technology
E-mail: tobias.uhlig@tu-dresden.de

Oliver Rose
Dresden University of Technology
E-mail: oliver.rose@tu-dresden.de

include selection for reproduction and selection for survival. Fueled by variation and selection, the fitness of individuals gradually improves with passing generations (see Fig.1). Obviously this also improves the results generated by the EA.

Natural selection is a mechanism for generating an exceedingly high degree of improbability. [2]

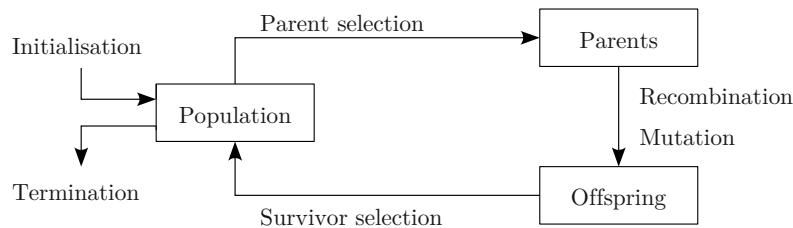


Fig. 1 Basic scheme of an evolutionary algorithm [1, p. 17].

Determining an adequate problem representation is the first step in designing an EA. This representation is called the genotype. Solutions to a given problem, called phenotypes are derived from the genotype. Consequently the quality of an EA depends very much on the chosen genotype.

Getting the representation right is one of the most difficult parts of designing a good evolutionary algorithm. Often this only comes with practice and good knowledge of the application domain. [1, p. 40]

A good problem representation supports the effective use of variation operators. Furthermore the optimal solution to a given problem should be represented in the resulting genotype space. However it is not necessary, that every possible phenotype may be encoded using a certain genotype. Various genotypes can be applied to a given problem, ranging from very abstract representations like binary encoding to more natural problem related ones, like permutation based encoding. For many problem instances it is very difficult to discern a single representation that is optimal to all intents and purposes. Especially the disruptiveness of the variance operator may differ for various representations. Since the prediction of the best- suited genotype is difficult, we propose an EA, relying on multiple genotypes, which compete with each other to generate the best solution. Individuals based on mutually incompatible representations are regarded as different species. In this multi-species evolutionary algorithm (MSEA) we expect that eventually the best representation will prevail and lead to the best results. Selection favors species, which have an inherent representation advantage since they should lead to fitter individuals. There already exist some implementations of MSEAs, but they do not employ different genotypes for different species [3]. Instead they artificially label individuals as certain species. This is done to avoid premature convergence by splitting the population into multiple subgroups.

The population based nature of EAs holds out much promise for identifying multiple optima, however, in practice the finite population size when coupled with recombination between any parents (known as panmictic mix) leads to the phenomenon known as genetic drift and eventual convergence around one optimum. [1, p. 155]

The diversification resulting from multiple subpopulations exploring different niches of a search space helps to avoid premature convergence. Subpopulations are generated using different species, which are restricted from mating. Alternatively races can be employed [5]. Individuals belonging to different races can mate with each other, but generally prefer mates of the same race. Consequently this leads to a certain degree of information exchange between subpopulations. Many other techniques have been developed to avoid genetic drift, for example aging of individuals . An introduction to them can be found in [1].

2 Scheduling Problems in Semiconductor Manufacturing

Semiconductor manufacturing has become one of the largest industries in the world. Hence there is a great interest in efficient production in this field. In addition to advancements in the fabrication process improving the operational process, should significantly increase the effectiveness of wafer fabrication and reduce the production costs. Scheduling is one important method to optimize the operational process. Today dispatching rules are commonly utilized in semiconductor manufacturing [6]. However, deterministic scheduling is a promising approach and ultimately may be necessary to further improve the effectiveness of wafer fabrication [7].

This paper will discuss the optimization of schedules for tool groups in a wafer fab. Oftentimes a single tool or tool group is the bottleneck in wafer fabrication. Usually these are very expensive tools operating at a high load. Optimizing the throughput of these bottlenecks will improve the performance of the whole fab and thereby reduce the costs of wafers. In semiconductor manufacturing there are basically three different types of tools: Simple tools processing one job at a time, batch tools and cluster tools. Batch tools can simultaneously process a number of parallel jobs. A group of jobs processed in parallel is called batch. In contrast, cluster tools also process parallel jobs, but each job is handled independently. In the context of semiconductor manufacturing, a job is a lot containing multiple wafers. The processing time of a job may depend on the number of wafers.

Regarding the three types of tools, we can identify some basic scheduling problems. These are sequencing, partitioning, and grouping of jobs. Sequencing deals with the optimal order of jobs, which is relevant for problems with sequence-dependent setup times or sequence-dependent process times. It is also very important for problems with customer oriented objective functions, that depend on delivery on time to satisfy certain due dates. The distribution of jobs to parallel machines is called partitioning. Scheduling for batch tools requires grouping of jobs, which are to be processed in a batch. These problems are usually very complex by themselves. However, most actual problems are a combination of more than one of them and therefore exhibit an even greater complexity. Scheduling problems are often categorized using the $\alpha|\beta|\gamma$ notation by Graham et al. [8]. The α field describes the machine environment, the β field represents special restrictions and the γ denotes the scheduling objective. According to that notation the problem discussed in this paper is:

$$R|lrc|Cmax$$

We face unrelated parallel tools (R) with sequence-dependent process times caused by different load port recipe combinations (lrc), which is typical for cluster tools. The scheduling objective is the makespan (Cmax). The makespan equates the completion time of the last job and therefore provides a measurement of the throughput. Other common scheduling objectives in semiconductor manufacturing are total weighted tardiness and maximum lateness.

The given problem is a special instance of Job Shop scheduling in combination with partitioning, which are both known to be NP-hard [9]. Hence, no algorithm is likely to be able to correctly solve them in an efficient way. Regarding partitioning we also consider dedication. Dedication is a constraint resulting from different machine qualifications. Jobs may only be processed on certain qualified machines.

2.1 Cluster Tools

Since the experiments performed for this paper employ cluster tools we introduce them briefly. For a more detailed introduction refer to [10] or [11]. Cluster tools are integrated tools employed in semiconductor manufacturing. A cluster tool is composed of multiple process chambers, arranged around a vacuum mainframe. It processes lots, which usually contain up to 25 wafers. Wafers can be processed in parallel in different process chambers (see Fig. 2). Effectively a cluster tool is an integrated job shop where the number of parallel jobs is limited by the load port count. Load ports are entry points for lots. A robot takes single wafers from a lot in a load port to a chamber where it is processed. A wafer is processed in one or more process chambers according to its recipe. Since different wafers usually compete for limited resources (available process chambers, shared transport robot) they mutually slow each other down. This effect can be modeled with slow down factors (SDF) for given recipe mixes. A SDF describes how much a job is slowed down in comparison to its raw processing time (RPT), which is the processing time a job would have, if it had exclusive access to the cluster tool. The estimated processing time of a job can be calculated as the product of SDF and RPT. Using a matrix prediction method based on SDFs and RPTs, we can predict the completion times for lots processed in a cluster tool [11]. This method updates the completion time of all active jobs whenever the load port recipe combination changes. This approach is significantly faster than a detailed simulation of a cluster tool and it is implemented in our simulator.

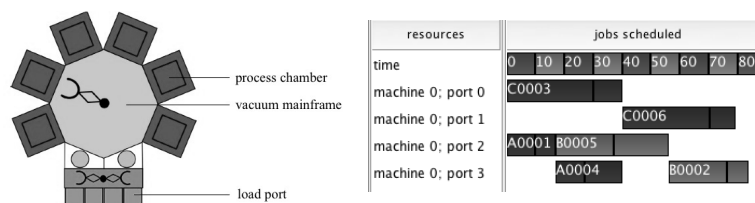


Fig. 2 A basic cluster tool with 4 load ports and a gantt chart of jobs processed on it.

3 Methods

3.1 On Schedules

With regard to our work, a schedule contains jobs, which are assigned to machines. In the context of semiconductor manufacturing, jobs are lots containing multiple wafers. The wafers of one lot are processed according to a certain recipe, which defines the necessary

sequence of processing steps. Therefore, the processing time of job on a certain machine depends on its recipe. In this paper, a job is denoted by a capital letter, different letters represent different recipes and an index may be used to identify the jobs. Jobs are placed in a job sequence. A basic schedule is an assignment of one job sequence to a single machine. We call this an atomic schedule. All other schedules are composed of atomic schedules. For example a schedule for a tool group with parallel machines consists of one atomic schedule for each machine. A job sequence is a permutation of jobs defining the order in which jobs should be processed on the assigned machine. Taking the job Sequence $A_1A_2A_3B_4B_5A_6$ as an example:

- A simple tool processes the jobs one at a time in the given order, first A_1 then A_2 and so on
- A batch tool removes multiple jobs from the sequence for each processing step, until a job with a different recipe occurs; the number is also limited by the batch size. With batch size = 2 the jobs are processed in this order $[A_1A_2][A_3][B_4B_5][A_6]$
- A cluster tool starts processing with the first n jobs (n = number of load ports), whenever a job finishes the next job is taken from the sequence

By mean of a simulation, a schedule is validated and the processing times for all jobs are calculated. Only a valid schedule may be further evaluated. A schedule may be invalid if for example a job was assigned to a machine, which is not qualified to process it.

3.2 The Scheduling Framework

Our framework was implemented to evaluate different metaheuristics for semiconductor tool scheduling. It consists of three main components: a simulator, schedule evaluation methods and scheduling algorithms. In general, the scheduling algorithms call the simulator to run their schedules. Subsequently they ascertain the quality of the generated solutions using the evaluation methods.

The simulation core is a very basic discrete event simulator. It was developed for rapid simulation of schedules. It lacks nearly all features of commercial simulators and just calculates the processing times of the jobs for a given schedule. It basically uses lookup tables assigned to machines to determine the processing time of a job on a machine depending on its recipe. Additionally, it supports tables with sequence dependent setup times for single tools and batch tools. Cluster tool behavior is simulated by means of the matrix prediction method [10] using special lookup tables containing slow down factors for all load port recipe combinations. The simulator can only simulate a schedule for a tool group with parallel tools. The supported tools are simple tools (one job at a time), cluster tools, and batch tools. At the moment it does not support sequential process steps and routing of jobs.

Evaluation methods can be applied to processed schedules to calculate different objective functions. Typical objective functions are makespan (C_{max}), maximum lateness (L_{max}), and total weighted tardiness (TWT). The framework currently supports all objective functions, which rely only on job attributes for calculation. Available job attributes include release dates, due dates, deadlines, weights, processing times, waiting times, and recipes.

The metaheuristics we employ are algorithms, which generate schedules for a given test setup. A test setup consists of a number of jobs and parallel tools belonging to one tool group. The resulting schedule contains ordered job sequences allocated to the machines. This paper discusses a modular evolutionary algorithm, which is introduced in the next paragraph. Other algorithms we implemented include simulated annealing, hill climbing, ant colony optimization, particle swarm optimization, and others.

3.3 A Modular Evolutionary Algorithm

To perform our experiments we developed a modular EA (see Fig. 3). In contrast to ordinary EAs, the algorithm is independent of the actual problem representation and works with any individuals, adhering to a given schema. The individuals are responsible for problem specific details. Selection methods are also designed modularly to allow easy replacement.

```
intitalize population
for number of generations do
  parents = Selector(population, n)
  for each parent do
    generate one offspring
    add offspring to children
  end for
  population = SurvivorSelector(population, children)
  store best solution and update statistics
  apply aging
end for
```

Fig. 3 Pseudo code of the modular evolutionary algorithm

A **Selector** chooses n parents from a given population. These parents are used to generate offspring representing new solutions. Usually the Selector favors fitter individuals to apply selective pressure. An individual may be chosen as parent more than once in each generation. The Selector may employ a stochastic or deterministic selection method. The stochastic methods we implemented include (among others) ranking selection, tournament selection and, fitness selection. An example for deterministic method is truncate selection.

The **SurvivorSelector** builds the next generation by selecting individuals from the current population and its children. In contrast to the reproduction selector an individual may only be chosen once for survival. This selection process may once again be stochastic or deterministic. In addition to fitness the SurvivorSelector may use other factors like elitism, age, race, or species for selection.

Resulting from the modular approach, we can easily adapt the selection strategy of our algorithm. This raises the responsibility for the experimenter setting up the algorithm since it increases the amount of parameters. Generally, an evolutionary algorithm works well if there is some kind of selective pressure in contrast to a purely random selection, but the intensity of selection affects the pace of evolution and the likeliness of premature convergence. A high selective pressure promotes the elitist individuals causing a very fast development. However, it also increases the likelihood of genetic drift towards a single local optimum. That is why it is important to utilize a well-balanced selection strategy to ensure a diverse population over time. A population consists of a number of unique individuals, which may belong to different species or races. The individuals we employ provide solutions to a given scheduling problem. This solution is their phenotype, generated depending on their genotype. The genotype is a problem dependent representation usually called DNA. Individuals of the same species share a common DNA representation. Individuals of the same race furthermore share a common generator logic for schedules and identical parameters. Races belonging to one species may differ in many characteristics but they remain compatible with regard to recombination. That said most of the times individuals of certain race have a strong bias to select an individual from the same race as mate.

Whenever an individual is selected as a parent it generates one offspring. Reproduction may be mitotic or meiotic. Using mitosis a genetically identical offspring is generated and a mutation operator is applied to its DNA. If meiosis is employed the individual uses its own Selector to chose a compatible mate from the current population. Subsequently, a new individual is generated using recombination and mutation operators. The most common recombination operator is crossover. Individuals are initialized at the beginning of the algorithm (initial population) or during their creation as offspring. During initialization a schedule is generated from the DNA. The resulting schedule is evaluated and a fitness value is assigned to the individual. The fitness value represents the quality of the generated schedule according to a given objective function.

To illustrate the concept of multiple species we will introduce two example species. The first species (S1) uses a text string to encode the genetic information. The DNA is a pattern containing recipes denoted as capital letters. To generate a schedule all Jobs are ordered according to their recipe to match this pattern (see Fig. 4). The species uses swap mutation and recombination, that is implemented as single point crossover, multi-point crossover, or uniform crossover. Obviously DNA generated with crossover may be incorrect since the number of certain recipes in the pattern may be wrong. To solve this problem we apply a correction algorithm. It replaces the first occurrence of a too frequent recipe with a recipe occurring too infrequently until the string pattern is valid. This correction is sometimes called a repair operator and can be seen as a special kind of mutation.

The resulting job sequence is cut into partitions of fixed size, which are approximately equisized. Recent studies have shown that this arbitrary constraint oftentimes leads to very good results [10]. This may be attributed to the reduced size of the search space and the common ground that all individuals can rely on. Especially regarding recombination different sized partitions may lead to significant disruption. However it is easy to find examples where equisized partitions are disadvantageous. For some problems there does not even exist a valid solution with equisized partitions.

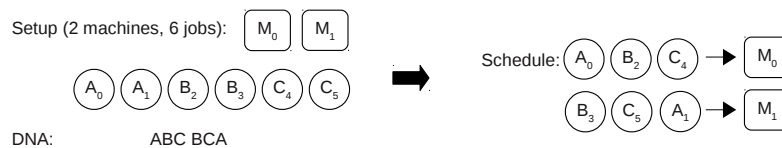


Fig. 4 Schedule encoding for species S1 using a recipe pattern and equisized partitions

The second species (S2) relies on flexible partition sizes. To achieve this we introduce special marker jobs as cutting points. For sequencing purposes these marker jobs are treated just like all other jobs. For this species we employ random number indices to encode the relative job position. The jobs are sorted according to their corresponding index number (see Fig. 5). We employ a stable sorting algorithm. This species employs crossover for recombination and supports two different mutation operators. Swap mutation exchanges two values in the DNA, while random resetting replaces one index with a new random value.

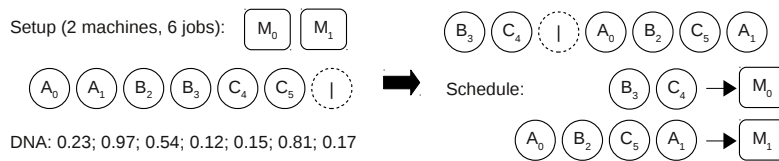


Fig. 5 Encoding of schedule used by species S2 employing a random number indeces. An additional dummy job is used to mark partitions (dashed circle).

Other species have been implemented as well, but will not be discussed in detail in this paper. Some of them use different approaches for partitioning employing a list containing machine numbers to explicitly assign jobs to certain machines. Furthermore there are combinations of methods introduced for S1 and S2. For example an individual using recipe patterns in conjunction with flexible partitions. The experiments performed for this paper only used S1 and S2.

3.4 Design of Experiments

For our experiments we employed five different test setups with parallel cluster tools (see Table 1). The tests include experiments with incompatible cluster tools, i.e., tools that cannot process jobs with certain recipes. This dedication of tools to certain jobs depending on given machine qualifications is a typical constraint in semiconductor manufacturing. Furthermore we have instances where a tool is significantly slower than other ones. As a reference algorithm we tested all problem instances with a Monte Carlo Method. This method returns the best solution of 10,000 randomly generated schedules. Each experiment has been repeated 100 times to minimize variance. For this paper, we evaluated a multi-species algorithm using two species as described above (S1 and S2). For reference purposes we also run simulations using the EA with only one species for each of them. For the evolutionary algorithms we used the following parameters:

- Fitness: Cmax as objective function
- 100 generations and 100 individuals per generation (equally distributed among species for multi-species instances) resulting in 10,000 simulation calls per run
- Parent selection: stochastic ranking selection
- Survivor selection: species-independent, deterministic with uncapped elitism
- Single point crossover for recombination

Table 1 Test Setups

No.	Jobs	Recipes	Machines
1	40	10	4 incompatible cluster tools with 4 load ports
2	30	5	2 incompatible cluster tools with 4/2 load ports
3	30	10	3 identical cluster tools with 4 load ports
4	40	10	3 identical cluster tools with 4 load ports
5	40	10	4 cluster tools with 4 load ports (1 slow, 2 incompatible)

All experiments were performed on a reasonably modern laptop. Calculation time for one algorithm run, with 10,000 simulation calls, was always well below 5 seconds for one repetition.

4 Results

The results of our experiments illustrate that all EAs perform significantly better than a Monte Carlo algorithm (see Table 2). On average the multi-species algorithm returned the best results. However, with regard to single tests it never was the winner. This can be attributed to the fact, that only two different species were employed and no special selection enforced the surviving of both species. Consequently there were at most two subpopulations exploring different niches. It can be assumed, that in most cases one species became extinct very soon, which effectively resulted in a single species EA. Nevertheless the multi-species proved to be very robust, since it found a solution for nearly all test runs. In contrast the EA using only S1 found no valid solution in 1 out of 10 cases while S2 found no solution in 1 out of 20 instances. Regarding the results it is obvious that S1 and S2 excel in different test cases. Generally S1 is more effective whenever sequencing is difficult and equisized partitions are not a hindrance. In contrast S2 excels at partitioning. The multi-species EA can benefit from both species, relying on the one, which is better suited to solve a certain task. Consequently it performed nearly as well as the better single species EA for all problem instances. For further reference we include the results of a simulated annealing algorithm. This algorithm returned slightly better but comparable results as the EA using species S1.

Table 2 Average improvement in tests 1-5 in comparison to Mont Carlo Method and percentage of successfully solved problems (100 repetitions per test)

	Evolutionary Algorithm with Individuals			Simulated Annealing
	S1	S2	S1 + S2	
Test 1	24.1%	21.9%	22.5%	25.6%
Test 2	9.0%	16.5%	15.9%	8.3%
Test 3	12.0%	11.2%	12.0%	13.6%
Test 4	19.0%	19.1%	18.2%	20.4%
Test 5	21.1%	16.9%	18.4%	23.1%
Avg. Gain	17.0%	16.9%	17.4%	18.2%
Solved Instances	90.0%	94.6%	99.6%	90.0%

5 Discussion

Evolutionary algorithms are general-purpose heuristics, which are often outperformed by specialized algorithms. Regarding cluster tool scheduling a simplex algorithm [12] is able to determine an optimal job sequence for many problems considering only a single tool with Cmax as the objective. However, this algorithm cannot be applied to problems with parallel tools. A promising idea would be an individual, which employs the simplex algorithm for sequence generation for cluster tools and uses a genetic code for partitioning. This would result in a hybrid EA, relying on domain knowledge. Similar approaches are often referred to as memetic algorithms [1, p. 173-185].

The algorithm introduced in this paper was developed for offline scheduling problems. However, in semiconductor manufacturing scheduling is often neglected in favor of simple dispatching rules, since the actual problem description may change very often. This for example can be caused by arrival of new jobs in the queue of a tool group. Furthermore there are unpredictable events like machine breakdowns, which occur very often in semiconductor manufacturing. These facts seem to imply that scheduling cannot be applied successfully to these problems. However, with regard to our fast simulator it is easy to completely

recalculate a new schedule in a few seconds, whenever a change occurs. Consequently the responsiveness of our proposed scheduling system should be adequate with respect to real world requirements.

At the moment our multi-species EA relies heavily on the idea that different implementations of individuals may have an inherent advantage with regard to different scheduling problems. However, future development might lead to an implementation, which outperforms other individuals for every problem instance. Consequently no further benefit would be expected if other inferior species were employed in conjunction with this super-individual. In this case one might resort to a multi-species algorithm employing a labeling approach instead of actual different implementations for speciation.

6 Conclusion

The multi species EA proposed in this paper achieved reliable, good results, although we employed only two species. The limited number of species prevented the algorithm from exploring multiple niches in the search space effectively. Hence it did not significantly improve the results with regard to specific experiments in comparison to conventional EAs. On average it returned the best results since it could always rely on the genetic representation which generated better results for a given problem instance. This behavior led to a very high robustness. Given a certain problem it is the best solution, if we do not know in advance which species is suited best to solve this problem instance. The MSEA eliminates the need to perform multiple EA runs to determine the best problem representation. It automatically comes up with the best one using the already existing selection operators of EAs.

In the future experiments using more species will be performed. Furthermore special selection mechanisms will be implemented to avoid early extinction of species. Hopefully this will lead to further improvements, by emphasizing the idea of exploring multiple niches. Additionally the idea of multiple species may be applied to multi-criteria problems, since different species can focus on different objectives.

References

1. A. E. Eiben and J. E. Smith, Introduction to Evolutionary Computing, Springer, Berlin Heidelberg (2007)
2. J. H. Bennett, Natural Selection, Heredity, and Eugenics, 27. Clarendon Press, Oxford (1983)
3. P. Husbans, Distributed Coevolutionary Genetic Algorithms for Multi-Criteria and Multi-Constraint Optimisation Evolutionary Computation, AISB Workshop, 150-165, Springer (1994)
4. F. Nouioua and A. Hamdi-Cherif, Using age, competition and dynamic parameters to control the populations diversity of an evolutionary algorithm, 432- 436. In INISTA 2009, (2009)
5. R. Conor, Racial Harmony and Function Optimization in Genetic Algorithms - The Races Genetic Algorithm, Evolutionary Programming Conference '95, 296-307 (1995)
6. A. Varadarajan, and S. C. Sarin, A survey of dispatching rules for operational control in wafer fabrication. In Information Control Problems in Manufacturing Vol. 1, Part 1. (2006)
7. M. Pfund, S. Mason, J. W. Fowler, 2006. Dispatching and Scheduling in Semiconductor Manufacturing: State of the Art and Survey of Needs, Handbook of Production Scheduling, 213-241. J. Herrmann, (eds.) (2006)
8. R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kann, Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey, 287-326. Annals of Mathematics 5, (1979)
9. M. Pinedo, Scheduling: Theory, Algorithm and Systems, Prentice Hall (2002)
10. T. Uhlig, Employing Evolutionary Algorithms to Minimize the Makespan of Schedules with Sequence-Dependent Process Times. Thesis, Department of Applied Computer Science, Dresden University of Technology. (2009)
11. R. Unbehaun and O. Rose, 2007. Predicting Cluster Tool Behavior with Slow Down Factors, 1755-1760. In Proceedings of the 2007 Winter Simulation Conference, (2007)
12. E. Boehl, Einsatz des Simplexverfahrens in einer Clustertoolanalyse. unpublished, (2008)

Case study: an effective genetic algorithm for the chemical batch production scheduling

Dmitriy Borodin • Bert Van Vreckem • Wim De Bruyn

Abstract The paper presents a case study – two genetic algorithms for solving the integer programming part of the MILP formulation for a particular chemical batch production scheduling problem. The standard MILP formulation of the batch production scheduling problem is decomposed into two sub problems: 1) unconstrained integer programming problem involving two independent assignment and one permutation vectors as solution representation and having one objective function; 2) constrained continuous linear program representing the original MILP formulation with integer variables fixed in terms of the solution of the first sub problem. The first sub problem is known as NP-hard and is difficult to solve by exact algorithms, the second sub problem is easy to solve in terms of exact methods. Two genetic algorithms are proposed to solve the first sub problem; the linear program acts as an objective function by delivering the solution of the original MILP model using the values of integer variables found by genetic algorithms. It is explained how the complete production schedule and the objective function value (the schedule cost) are deduced. The genetic algorithms combine the approaches for solving both assignment and permutation sequencing problems. The solution representations, algorithms' convergence and parameters are discussed. The efficiency of both algorithms is proved by the computational results that are compared with the optimal solutions and with another heuristic technique.

1 Introduction

Scheduling is a very important issue in operations research. Despite many successful researches, books, case studies and reviews on various scheduling problems, the effective optimization techniques for real industry scheduling problems are still of demand.

Dmitriy Borodin
Production Information Systems Lab, University College Ghent
E-mail: Dmitriy.Borodin@hogent.be

Bert Van Vreckem
Production Information Systems Lab, University College Ghent
E-mail: Bert.VanVreckem@hogent.be

Wim De Bruyn
Production Information Systems Lab, University College Ghent
E-mail: Wim.DeBruyn@hogent.be

This paper deals with the scheduling of a chemical batch process. Batch production scheduling problems have been actively studied: the recent reviews [11] and [12] provide various problem formulations and solution approaches, papers [8] and [13] give the overview of the recently developed optimization techniques for the short-term batch production scheduling and provide a case study, which is similar to the present paper in some sense.

The MILP problem formulation ideas, used in this paper, are similar to the ones, explained in [1], [2] and [14]. The idea of the problem reduction was also used in the mentioned papers. The paper mainly focuses on the genetic algorithm, at the same time explaining the necessary problem formulation and reduction details and issues.

Genetic algorithms, among other heuristic and meta-heuristic optimization techniques, were widely and successfully used for solving scheduling problems, and chemical batch production problems in particular [1], [2], [7], [13], [16]. The contribution of this paper is in applying the genetic algorithm to the problem with the solution representation, which was not researched in other papers and, thus, is not typical for the genetic algorithms. The promising computational results help to conclude that the proposed genetic algorithms can be recommended for a generic class of problems, similar to this case study.

The rest of the paper is structured as follows. Section 2 provides the problem insight. In subsection 2.1, the problem background is described. Then, subsection 2.2 gives some information on the mixed-integer formulation of the master problem without going into detail; in subsection 2.3 we explain how the problem is converted into the problem with one objective function but independent solution vectors; the evaluation function is introduced. In Section 3 we provide a short introduction to genetic algorithms, present two different solution representations and two solution approaches, describe the steps and parameters of the proposed genetic algorithms. Section 4 presents numerical results, discussion on different solution representations and genetic algorithm parameters, and comparison of the genetic algorithms efficiency with exact optimal solutions and solutions, obtained by using the Kernighan-Lin heuristic (a variation of a tabu search). Section 5 concludes the work and proposes the statements for the future research.

2 Problem statement

For the problem under study, two mixed-integer programming models were developed so far: 1) with discrete representation of time horizon (one time unit was equal to 8 hours); 2) with continuous representation of time horizon. As the second model reflected the industry needs better and, in fact, was closer to reality, the efforts were concentrated on solving this model. As this MILP model was NP-hard, the problem was decomposed into two sub problems. This decomposition was motivated by the idea of separating integer and continuous variables as continuous linear programs are easy to solve. Thus, the MILP-model is decomposed into the following sub models: 1) the unconstrained integer programming model; 2) the constrained linear program. The linear program represents the original MILP-model but with integer variables fixed by solving the integer programming part. In order to describe this approach formally, a so-called evaluation function (EV) involving three independent sub vectors but one objective function (the schedule cost) was introduced. This work was presented, discussed and got positive feedback at the 24th European conference on Operations Research in Lisbon in 2010 (we refer to the abstract published in [5]) and is explained in details in the yet unpublished paper [6]. Reference [6] is expected to appear in one of the European journals within a reasonable time period. It contains the optimal solutions for the problem instances (computed using the commercial solver CPLEX) as well as the Kernighan-Lin heuristic results. These give us an opportunity to evaluate the efficiency of the developed genetic algorithms.

Below we present the problem background, the continuous time formulation from [6] in the detail enough to make it possible to understand the initial problem complexity; we also provide the EV description for the reader to make it easier to understand the integer programming sub problem which the genetic algorithms were intended to solve.

2.1 Problem background

The production unit under study is responsible for the production of various enzymes.

The production process of an enzyme involves 4 main steps:

- the fermentation, where the enzyme is produced by living microorganisms;
- the recovery, where the enzyme is purified mainly by filtration;
- the formulation, where the enzyme is stabilized and standardized in form of liquid or granules;
- the packaging, where the enzyme is put in containers or drums to be stored and transported to the customers.

The present work focuses on scheduling production for the first two steps (fermentation and recovery) of this process, as they represent the bottleneck stages of the production process.

The fermentation consists in multiplying an initially small amount of microorganisms and making them produce a desired enzyme. The first step is the seed fermentation: the microorganisms are placed in a small tank where they start growing and reproducing. After a limited time (typically 8 to 24 hours), the cells are transferred in a larger tank, a main fermenter, containing previously sterilized raw materials (nutrients) and water. During the main fermentation, the cells continue to grow and multiply, while producing the enzyme. The duration of this second step is significantly longer than the first one, typically 30 to 300h, depending on the type of enzyme considered.

The following production stages are used to produce an enzyme:

- seed fermentation;
- main fermentation;
- broth preparation in the buffer tanks;
- recovery.

To perform these operations, the case study factory possesses the following resources:

- three identical seed fermenters, $S = 2$;
- five identical main fermenters, $M = 5$;
- four buffer tanks: three identical small, $B^s = 3$, and one large, $B^l = 1$ ($B^s + B^l = 4$);
- one recovery line, $R = 1$.

A number of batches are to be processed over the planning horizon (which is in fact defined by customer demand). Several enzyme types are produced by the factory, denoted $p \in \{1 \dots P\}$. We have to produce α_p number of batches per each enzyme type, the size is given by q_p . Each batch is indexed by (p, j) consequently, where $p \in \{1 \dots P\}$ and $j \in \{1 \dots \alpha_p\}$.

Both seed and main fermentation phases include three operations: 1) the setup or preparation of the fermenters; 2) the fermentation itself (duration is denoted as π_p^S and π_p^M respectively); 3) the tear down or cleaning of the fermenters. So, the total time required for a batch to be processed either in a seed or in a main fermenter is given by sums of the times necessary to perform these three operations. Moreover, there are operational constraints: 1) a batch should be transferred without any delay to a main fermenter (which must be already setup or prepared) once the seed fermentation is finished (i.e. before starting the tear down or cleaning operations on the seed fermenter); 2) a batch should be transferred to the buffer tanks without any delay, once the main fermentation is finished (i.e. before starting the tear down or cleaning operations on the main fermenter); 3) the setup of a main fermenter requires a number of difficult manual operations, and because of the manpower restrictions at most one batch can be setup in main fermentation within a production shift.

When the fermentation is completed, the mixture of cells, nutrients and enzymes, called the broth, is transferred into a buffer tank where it is prepared for the recovery operations. Then the recovery takes place: it consists in separating the desired enzyme from the broth, using filtration and purification. The duration of the recovery phase is denoted δ_p^R . During the recovery phase, a batch is progressively transferred from a buffer tank(s) to the filtration unit. That is why the buffer tank(s) in which the batch is stored remains occupied until the filtration of the entire batch is finished.

An important issue for the broth preparation phase is a batch size. The batches may have two sizes – big or small, i.e. $q_p \in \{1,2\}$. A small batch can be processed either by a large or a small buffer tank while a big batch can be processed by two small or one big buffer tanks; exactly one batch can be stored in a buffer tank at the same time. Once a big or a small batch has been transferred into one or two buffer tanks, it has to be processed in the δ_p^R amount of time, but cannot be stored longer than Δ_p^B , i.e. the total time spent by a batch in the buffer tanks before filtration must be in the interval $[\delta_p^B, \Delta_p^B]$.

In case a big batch is stored in two small buffer tanks, one of these tanks is emptied after half of the batch has been filtered, i.e. it is free after $\frac{\delta_p^B}{2}$ hours, and thus can be used to store another batch.

We denote $DD_{p,j}$ (the due date) the date when the batch should have been processed through both fermentation and recovery stages.

Figure 1 provides the graphical overview of the production stages.

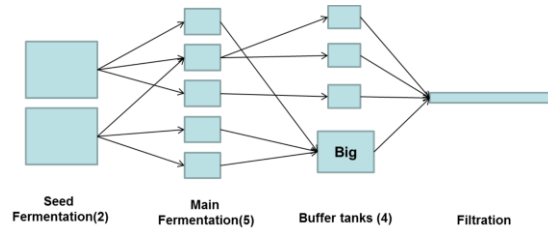


Figure 1. Production process stages.

2.2 Continuous time formulation information.

The following continuous decision variables are defined:

- $t_{p,j}^M$ is the starting time for the main fermentation of batch (p,j) , which means the moment when batch (p,j) goes from a seed fermenter to a main fermenter;
- $t_{p,j}^R$ is the starting time for the recovery of batch (p,j) .

The discrete binary variables are introduced to determine, for each production step, which resource will process each batch, i.e. to represent resource assignment:

- $sa_{p,j}^s = 1$ if batch (p,j) is processed by seed fermenter s , 0 otherwise;
- $ma_{p,j}^m = 1$ if batch (p,j) is processed by main fermenter m , 0 otherwise;
- $ba_{p,j}^b = 1$ if batch (p,j) is processed by buffer tank b , 0 otherwise;

The production sequence on each resource is determined by the following discrete binary variables:

- $sps_{p,j,q,k}^s = 1$ if batch (p,j) is the predecessor of batch (q,k) on seed fermenter s , 0 otherwise;
- $mps_{p,j,q,k}^m = 1$ if batch (p,j) is the predecessor of batch (q,k) on main fermenter m , 0 otherwise;
- $bps_{p,j,q,k}^b = 1$ if batch (p,j) is the predecessor of batch (q,k) on buffer tank b , 0 otherwise;
- $rps_{p,j,q,k}^b = 1$ if batch (p,j) is the predecessor of batch (q,k) on the recovery line, 0 otherwise;
- $mpps_{p,j,q,k} = 1$ if main fermentation of batch (p,j) begins right before the main fermentation of batch (q,k) , 0 otherwise.

Due to manpower restrictions, there must be a certain delay between the setup of two main fermenters. Variables $mps_{p,j,q,k}^m$ do not provide the information about the order of two batches that are not processed by the same fermenter that is why variables $mpps_{p,j,q,k}$ were introduced; they define the overall sequence for the main fermentation.

There were a few objective functions introduced for the problem: 1) minimize the makes plan; 2) minimize the total weighted earliness-tardiness, 3) minimize the mean flow time. As a criterion 2 suits the factory's requirements, it is used further for all computational experiments as an objective function:

$$\text{Minimize } \sum_{p=1}^P \sum_{j=1}^{\alpha_p} \left(\pi_{p,j}^+ [t_{p,j}^R + \delta_p^R - DD_{p,j}]^+ + \pi_{p,j}^- [t_{p,j}^R + \delta_p^R - DD_{p,j}]^- \right) \quad (1)$$

where $[x^+] = \max(x, 0)$, $[x^-] = -\min(x, 0)$.

It is worth pointing out that, if the values of all discrete variables are fixed, there is only a continuous linear program left to solve. In other words, if we find the production sequence on each resource of the different production levels (i.e. seed fermentation, main fermentation, buffer tanks and recovery line) then the optimal solution can be found by solving a continuous linear program, which is of course much faster than solving a mixed integer linear program.

This idea leads to the following conclusion: the complete schedule and the objective function value can be deduced from the production sequence on the recovery line, the assignment of all batches to the main fermenters and on the assignment of all batches to the buffer tanks.

The next section describes this approach in detail. It is called the evaluation function, as it computes the schedule and its cost by providing the values of the mentioned above sequences and assignments. It formulates a problem to be solved with a heuristic. We developed the genetic algorithm for this purpose; the algorithm is presented in section 4.

2.3 Evaluation function

The aim of this approach is to represent a solution of the problem with the three following vectors:

- the production sequence on the recovery line
 $\Sigma^R[i] = (p, j)$ means that batch (p, j) is the i^{th} batch to be processed on the recovery line;
- the assignment of all batches to main fermenters
 $A^M[m]$ is the subset of batches that are assigned to main fermenter m (represented by a linked list);
- the assignment of all batches to buffer tanks
 $A^B[b]$ is the subset of batches that are assigned to buffer tank b (represented by a linked list).

As it was already pointed out, it is possible to deduce a complete schedule using only these three vectors. The point is then to find the value of the vectors that will give the best schedule. Below we provide the details on how to compute the evaluation function, i.e. the function which, given the three vectors described above, determines the complete schedule and its associated cost.

Suppose that the values of all binary variables of the continuous-time formulation have been fixed. Then the remaining problem is a continuous linear program: indeed, the only remaining variables of the problem are continuous.

Now let Σ_0^R , A_0^M and A_0^B be the current value of the three input vectors. They actually provide all necessary information about the sequence on the main fermenters, on the buffer tanks and on the recovery line. Indeed, we know that if batches (p, j) and (q, k) are on the same main fermenter m (the information is provided by A^M) then (q, k) will be a successor of (p, j) on m if and only if (q, k) is processed after (p, j) on the recovery line. The same is true

for the sequence on the buffer tanks. So, having Σ_0^R , A_0^M and A_0^B it is possible to compute the values of variables ma , ba , mps , bps and rps .

However, we do not have any information regarding the production sequence on seed fermenters or the global sequence on main fermenters (necessary to fix the values of variables sa , sps and $mpps$). This is the reason why there are two steps to obtain the complete schedule.

In the first step, a preliminary schedule is obtained by solving the linear program without taking into account the constraints regarding seed fermenters or manpower sequences. This schedule gives a production sequence for the main fermentation and one for the seed fermentation. Those sequences are then used to set the values of variables sa , sps and $mpps$. In the second step, now that all binary variables have fixed values, the complete schedule can be computed by solving the whole linear program. The total tardiness is the value of the evaluation function EV for point $(\Sigma_0^R, A_0^M, A_0^B)$.

With the evaluation function, it is possible to use a meta-heuristic algorithm to explore the solution space (defined by all possible values for vectors Σ^R , A^M and A^B) to find the best possible schedule. Section 3 proposes the genetic algorithm for this purpose. The next subsection expands the mentioned three vectors and does some transformations in order to represent the problem better for the genetic algorithm; the problem for the genetic algorithm is also stated in the subsection.

2.4 Problem statement for the genetic algorithm

Let us consider the following problem:

Find the values Σ_0^R , A_0^M and A_0^B giving the minimum to EV.

As the instance data we have:

- the number of batches α_p for each enzyme type p .
As already described in section 2.1, each batch is indexed by (p, j) consequently, where $p \in \{1 \dots P\}$ and $j \in \{1 \dots \alpha_p\}$.
- $DD_{p,j}$ – due date for batch (p, j) .
The date when the batch should have been processed through both fermentation and recovery stages.

Further, in order to represent the batches as one list, we add them to the list consequently, for each p , as follows:

$$b_{list} = (\{1\}, \{2\}, \dots, \{P\}), \quad (2)$$

where each subset has $|\alpha_1|, |\alpha_2|, \dots, |\alpha_P|$ equal elements respectively.

For instance, $P = 3$, $\alpha_1 = 2$, $\alpha_2 = 3$ and $\alpha_3 = 4$.

Then the batch list is the following:

$$(1, 1, 2, 2, 2, 3, 3, 3, 3) \quad (3)$$

As each batch is unique, we enumerate the list:

$$b_{sequence} = \left(1, 2, \dots, \sum_p \alpha_p \right) \quad (4)$$

The enumeration list for example (3):

$$(1, 2, 3, 4, 5, 6, 7, 8, 9)$$

Thus, each element of list (4) expresses the place of the respective batch in the sequence, and the list itself is a sequence; bijection between b_{list} and $b_{sequence}$ has been defined.

Considering these operations, we assume that as an input data we have the list of batches (2) and the due dates respectively listed for each batch.

Now we expand the three vectors in terms of the defined bijection.

$$A^M = (\{b_{sequence}^M[m_1]\}, \{b_{sequence}^M[m_2]\}, \dots, \{b_{sequence}^M[m_5]\}), \quad (5)$$

where $b_{sequence}^M[m_i]$ is a sequence of batches assigned to main fermenter i

$$A^B = (\{b_{sequence}^B[b_1]\}, \{b_{sequence}^B[b_2]\}, \dots, \{b_{sequence}^B[b_4]\}), \quad (6)$$

where $b_{sequence}^B[b_i]$ is a sequence of batches assigned to buffer tank i

$$\Sigma^R = (\{b_{sequence}^R\}), \quad (7)$$

where $b_{sequence}^R$ is an ordered sequence of batches on the recovery line

Representation of (5) and (6) is similar to an assignment problem solution representation and (7) is a permutation representation.

Three vectors for example (3) may have the following values:

$$A_1^M = (\{3,5,2\}, \{1,7\}, \{9\}, \{6\}, \{4,8\}), \quad (8)$$

where $b_{sequence}^M[1] = \{3,5,2\}$, $b_{sequence}^M[2] = \{1,7\}$, $b_{sequence}^M[3] = \{9\}$,
 $b_{sequence}^M[4] = \{6\}$ and $b_{sequence}^M[5] = \{4,8\}$ are sequences of batches

assigned to main fermenters 1 to 5 respectively

$$A_1^B = (\{1,4,8\}, \{9,2\}, \{3,5\}, \{6,7\}), \quad (9)$$

where $b_{sequence}^B[1] = \{1,4,8\}$, $b_{sequence}^B[2] = \{9,2\}$, $b_{sequence}^B[3] = \{3,5\}$ and
 $b_{sequence}^B[4] = \{6,7\}$ are sequences of batches assigned to

buffer tanks 1 to 4 respectively

$$\Sigma_1^R = (\{2,5,6,3,9,7,1,8,4\}), \quad (10)$$

where $b_{sequence}^R = \{2,5,6,3,9,7,1,8,4\}$ is an ordered sequence of batches on the
recovery line.

Finally, we define the problem to be solved by the genetic algorithm:

Find the values A_0^M, A_0^B, Σ_0^R such that $EV(A_0^M, A_0^B, \Sigma_0^R, b_{list}, DD_{p,j}) = \min_{A^M, A^B, \Sigma^R} EV$.

In the next section, we propose two solution encodings for (5) and (6) to be used in a genetic algorithm; (7) remains a permutation in both cases.

3 Genetic algorithm

First, we provide a short introduction into genetic algorithms.

Genetic algorithm (GA) is a class of meta-heuristic algorithms, involving the population-based search technique with the “survival of the fittest” principle. After the initial set of solutions (called population) is generated (randomly or using a solution construction method), the series of steps (selection, crossover, mutation) is repeated consequently until a stopping criteria is satisfied [3]. The selection step is responsible for selecting some “good” solutions for the crossover and mutation. After the new solutions have been created, they are evaluated and selected to be included into the next population. A crossover operator generates new solutions from a number of already existing solutions (parent solutions or parents, usually only two parents are involved) hopefully taking good features from them, while mutation enhances the diversity and provides an opportunity to escape from the local optima.

GA is widely used for scheduling problems with both binary and permutation solution representation, providing good results [1], [7], [11]-[16]. GA was used for efficiently solving assignment problems as well. We propose the algorithm that combines the solution of permutation sequencing and two independent assignment problems but with one and the same objective function.

3.1 Two solution representations

As in most papers the genetic algorithm operates with solutions represented as strings of variables (one dimensional vector) with fixed length [11], [12], and the sub vectors $b_{sequence}[m_i]$ and $b_{sequence}[b_i]$ have variable length, we decided to convert representation (5)-(6) to satisfy this condition.

The idea of the first representation is motivated by the similarity of (5)-(6) to a permutation representation, but separated in some points. Thus, (5) and (6) can be represented as the following vectors respectively:

$$\begin{aligned} A_{sequence}^M &= (\{b_{sequence}^M\}) \\ A_{assignment}^M &= (\{m_0, m_1, m_2, m_3, m_4, m_5\}) \end{aligned} \quad (11)$$

where $b_{sequence}^M$ is an ordered sequence of batches and m_i means that batches $\{b_{sequence}^M(m_{i-1}, m_i)\}$ are assigned to main fermenter i ($i = 1, 2, \dots, 5$), $m_0 = 0$, $m_5 = |b_{sequence}^M|$.

$$\begin{aligned} A_{sequence}^B &= (\{b_{sequence}^B\}) \\ A_{assignment}^B &= (\{b_0, b_1, b_2, b_3, b_4\}) \end{aligned} \quad (12)$$

where $b_{sequence}^B$ is an ordered sequence of batches and b_i means that batches $\{b_{sequence}^B(b_{i-1}, b_i)\}$ are assigned to buffer tank i ($i = 1, 2, \dots, 4$), $b_0 = 1$, $b_4 = |b_{sequence}^B|$.

As an example we represent (8) in the form of (11):

$$\begin{aligned} A_{sequence}^M &= (\{3, 5, 2, 1, 7, 9, 6, 4, 8\}) \\ A_{assignment}^M &= (\{0, 3, 5, 6, 7, 9\}) \end{aligned} \quad (13)$$

In this case the solution is represented as five vectors with fixed length.

The second representation is motivated by an assignment problem representation for the genetic algorithm. (5)-(6) are represented as follows:

$$\begin{aligned} A_{sequence}^M &= (\{b_{sequence}^M\}_{sorted}) \\ A_{assignment}^M &= (\{mf_1, mf_2, \dots, mf_N\}) \end{aligned} \quad (14)$$

where $\{b_{sequence}^M\}_{sorted}$ is a sequence of batches sorted in the ascending order and $mf_i \in [1, 2, \dots, 5]$ is the number of the main fermenter the i^{th} batch goes to, $N = |b_{sequence}^M|$

$$\begin{aligned} A_{sequence}^B &= (\{b_{sequence}^B\}) \\ A_{assignment}^B &= (\{bt_1, bt_2, \dots, bt_N\}) \end{aligned} \quad (15)$$

where $\{b_{sequence}^B\}_{sorted}$ is a sequence of batches sorted in the ascending order and $bt_i \in [1, 2, \dots, 4]$ is the number of the batch the i^{th} batch goes to, $N = |b_{sequence}^B|$

Again, we represent (8) in the form of (14):

$$\begin{aligned} A_{sequence}^M &= (\{1, 2, 3, 4, 5, 6, 7, 8, 9\}) \\ A_{assignment}^M &= (\{2, 1, 1, 4, 1, 5, 2, 5, 3\}) \end{aligned} \quad (16)$$

We assume that the sorted vectors $A_{sequence}^M$ and $A_{sequence}^B$ are equal and constant for a certain problem instance and only three vectors are considered as variable. So, in case of (14)-(15) the solution is represented in three variable vectors with fixed length.

Note. Both duplicates and missing values are acceptable for the assignment parts in (11)-(12) and (14)-(15). The meaning of a duplicate (as well as of a missing value) is that not all main fermenters or buffer tanks must process batches; some of them may process no batches. For example, such situation is possible when the number of batches to be scheduled is less than the number of main fermenters available (3 batches and 5 main fermenters). This note makes it possible to use genetic operators developed for binary solution representations.

3.2 Two solution approaches

As there are a few independent vectors in both solution representations, we propose two approaches to be used in terms of the genetic algorithm.

The idea of the first approach is that all solution vectors are modified simultaneously, i.e. crossover and mutation operators are applied to all sub vectors, in terms of one iteration; we call this approach simultaneous.

In the second approach, only one vector is modified within one iteration, the values of other vectors are fixed as constants; we call this approach consequent.

3.3 Genetic algorithm

The proposed genetic algorithm is similar for both solution representations (the difference is in number of vectors, three or five). Genetic operators used in the algorithms are described in the next sub section.

Step 0 and step 1 are similar for both approaches.

Step 0. To distinguish two solution approaches, we introduce the vector of binary variables $sv = (\{v_1, v_2, \dots, v_5\})$, where $v_i \in \{0,1\}$ determines if the i^{th} vector of the solution representation is modified within the current iteration ($v_i = 1$) or not ($v_i = 0$). In solution representation (11)-(12), (7) all v_i can participate, and in representation (14)-(15), (7) v_1 and v_2 are always fixed and thus are equal to 0. So, when we speak of all v_i , we mean all v_i sufficient for the current solution representation.

Step 1. Let all $v_i = 1$. Generate an initial population $P(0)$ of size P_s with randomly generated solutions; evaluate them by calculating the EV value for each solution. Let $k = 0$, k denotes an iteration number.

The procedure of the GA for the simultaneous approach is described as follows.

Step 2. We denote the best solution of population $P(k)$ as $EV_0(k)$ meaning both the solution itself and the value of the evaluation function EV . If a stopping criteria is satisfied (the number of iterations), return $EV_0(k)$ and the respective complete schedule of the master problem. Otherwise, continue to the next step.

Step 3. Select pairs of solutions for crossover from population $P(k)$; the number of pairs varies depending on the selection method. Perform crossover (the number of times depends on the selection and crossover method, but the average is $P_s/2$ times) and mutation (according to the mutation probability) operations to form a temporary population, after that select solutions from this temporary population to form a new population $P(k + 1)$, then let $k = k + 1$ and return to step 2. The solutions to be included into the new population are selected according to the chosen selection method, as described in the following sub section.

The procedure of the GA for the consequent approach is described as follows.

Step 2. As k denotes a global iteration, let the best solution of iteration k be $EV_0(k)$. If a stopping criteria is satisfied (the number of global iterations), return $EV_0(k)$ and the respective complete schedule of the master problem. Otherwise, continue to the next step.

Step 3. Let $i = 1$ for representation (11)-(12), (7) or $i = 3$ for representation (14)-(15), (7).

a. Set $v_i = 1$ for $i = 1$ or $i = 3$ and all other $v_{\{1,2,\dots,5\} \setminus i} = 0$. We denote kv as a number of iterations for the sub vectors. Let $kv = 1$. For these iterations, vector v_i is variable and the values of the other vectors $v_{\{1,2,\dots,5\} \setminus i}$ are fixed and equal to the best solution of iteration $kv - 1$; if $kv = 1$, then the values from the previous global iteration k are used.

b. Select pairs of solutions for crossover from population $P(k)$; the number of pairs varies depending on the selection method. Perform crossover (the number of times depends on the selection and crossover method, but the average is $P_s/2$ times) and mutation (according to the mutation probability) operations to form a temporary population, after that select solutions from this temporary to form a new population $P(kv + 1)$, then let $kv = kv + 1$; if the stopping criteria of the sub algorithm is satisfied (number of sub iterations) and 1) $i = 5$ (the last sub vector was processed), then return the best $EV(kv)_0$ and go to step 2; 2) $i \neq 5$, set $i = i + 1$ and go back to sub step a.

In other words, for the simultaneous approach all genetic operations are repeated for a certain number of iterations. For the consequent approach, a global iteration executes the genetic operators consequently for each sub vector while the values of other sub vectors are constant (fixed) and used in the next population in the unchanged form; thus, each global iteration has a best solution value, its meaning is similar to the best value of each iteration of the simultaneous approach.

In case of the three vector representation step 3b executes three series of sub iterations, and in case of five vectors – five series.

3.4 Implementation of the genetic algorithm

The solution representations involve two types of vector representation: 1) permutations which are widely used in scheduling problems and in the travelling salesman problem; 2) vectors containing integer numbers, these vectors are similar to the binary encoding in the sense of genetic operators and are popular in scheduling and assignment problems. Thus, such representations make it possible to use the best practices of the mentioned problems solutions.

As crossover operators, we used partially-matched (PMX), cyclic (CX) and order (OX) crossovers for the permutation vectors. We refer to [3], [9] for the details about these crossovers. For other vectors the one- and two-point and uniform crossovers were employed. Reference [9] describes these operators and analyzes theoretical difference between them.

As we didn't notice any significant difference between these operators for our problem instances, at each iteration the crossover operator was picked up randomly from these lists. It provides an opportunity to enhance the exploitation ability of the search.

Regarding the mutation, we used either the swap (two random elements are swapped) or insert (a random element is removed from one location and inserted into another) mutation for all vectors, for more details please see [9]. The mutation probability was set to 0.5, as it provided the best result.

For the selection method, first we implemented and tested the most popular ones: roulette-wheel, tournament and Genitor-Whitley [4]. But because of the problem specific feature – when the evaluation function wasn't able to deduce a feasible schedule, the EV was given a big value (1000 000) – these methods didn't provide good results. Then, we applied the two-random, cascade and “all with best” methods.

The two-random method randomly selects two solutions for the crossover and removes them from the population, until the population is empty. In this case the population size P_s must be even. In the cascade selection the solutions in the population are sorted in the ascending order by the EV value, then i and $i + 1$ solutions are selected for the crossover. “All with best selection” crosses over every solution with the best solution of the population; this method provided good results for some problems [15], but wasn't efficient in our case. The best results were obtained by using the two-random and cascade methods. In our implementation, on each global iteration the selection method is chosen randomly from these two methods.

In both methods, the temporary population is created from both original and crossed over solutions (parents and children respectively), and then the population is sorted in the ascending order and is cut from the “top” to fit the defined population size. Thus, the elitism strategy is “automatically” implemented, i.e. the best solution of the current population migrates to the next population.

It is also important to take the size of batches into account. As it is described in section 2.1, two sizes are possible, and size matters only for the broth preparation phase when a batch is processed in buffer tanks. Moreover, only the situation when the big batch is assigned to small buffer tanks requires special processing. Truly, if a big or a small batch is assigned to the large buffer tank, it is processed as usual. In other words, the attention must be paid to batch assignment in vector (6): $A^B = (\{b_{sequence}^B[b_1]\}, \{b_{sequence}^B[b_2]\}, \dots, \{b_{sequence}^B[b_4]\})$. To differ the sizes in all proposed GA and solution representations, the following algorithm is used. Once the batches are assigned, i.e. vector (6) got its values, the sizes of the batches assigned to buffer tanks 1 to 3 are checked. If a batch is of the big size, then it is split into two batches of the same type. One of these two new batches remains in the same buffer tank, another one is randomly assigned to one of the two remained small tanks. This algorithm is explained by the production realities and rules, i.e. in such a case the second half of a big batch is processed only in another small tank and not in a large one. The deterministic choice of the buffer tank to process the second half of a big batch was tested (the tank with smaller number of assigned batches were chosen) but it resulted in the restriction of the search space, that is why the random character of batch choice is used.

4 Numerical results and analysis

In this section, we present the numerical results of the computational experiments for 11 problem instances involving 10, 15, 20 and 30 batches. The results are compared with the optimal solutions (presented in [6] and calculated using the CPLEX solver) as well as with the Kernighan-Lin (KL) solutions (also presented in [6]). The proposed above solution representations and solution approaches are compared and discussed.

The algorithm was coded in C# using the Microsoft Visual Studio 2005 .NET environment, using the same evaluation function program as in [6]. The computations were performed on the PC equipped with the Intel Core 2 duo 2.4 Mhz processor and 4 Gb of memory. The experiments of [6] were run on a comparable PC, so it is possible to compare the CPU times as well.

Table I represents the overall comparison results. The CPLEX best column contains the optimal solutions, KL best – the Kernighan-Lin heuristic results (obtained in the KL time) and GA best – the GA results (computed in GA time). SA denotes the Solution Approach used to deliver the result shown in the GA best column (S denotes simultaneous and C denotes consequent), and SR – Solution Representation (1 is for representation (11)-(12), (7), 2 – for (14)-(15), (7)). This table shows that the GA outperforms the KL in most cases. Also it is obvious that the consequent approach with the second solution representation (C2) provides the solutions of better quality.

TABLE I. OVERALL COMPARISON TABLE

Instance	CPLEX best	KL best	KL time	GA best	GA time	SA	SR
N10_1	90	91	7	93	5	S,C	1,2
N10_2	30	35	16	30	17	S,C	1,2
N10_3	42	56	10	44	6	C	2
N10_4	49	52	14	50	5	C	2
N10_5	43	48	6	45	12	C	2
N15_1	73	77	80	76	25	C	2
N15_2	43	45	112	45	34	C	2
N15_3	57	70	39	66	75	C	2
N20_1	52	54	490	54	180	C	2
N20_2	58	66	260	64	194	C	2
N30_1	—	180	1304	186	1560	C	2

Table II shows the results of two solution approaches and two solution representations. Like in table I, C denotes consequent, S – simultaneous, and 1 or 2 - solution representation (11)-(12), (7) or (14)-(15), (7) respectively. Though the C2 approach is advantageous, other combinations also produce the solutions of good quality within the reasonable time period, and also can be useful, as it is suggested in the next section.

TABLE II. GA APPROACHES AND SOLUTION REPRESENTATIONS COMPARISON

Instance	C1	C2	S1	S2
N10_1	93	93	93	93
N10_2	30	30	30	30
N10_3	48	44	52	49
N10_4	52	50	56	54
N10_5	49	45	51	49
N15_1	79	76	84	82
N15_2	51	45	54	52
N15_3	69	66	71	69
N20_1	60	54	70	64
N20_2	70	64	74	70
N30_1	201	186	215	205

The result of the GA work (both solution quality and computational time) depends on the algorithm parameters. Trying to keep the stochastic feature of the algorithm, we used random

choice of the algorithm parameters on different iterations (thus enhancing the exploration and exploitation algorithm abilities).

TABLE III. THE C2 GA PARAMETERS

Instance	Population size	Iterations	Global iterations	GA best
N10_1	600	40	2	93
N10_2	100	50	1	30
N10_3	300	40	2	44
N10_4	100	40	1	50
N10_5	100	40	1	45
N15_1	100	100	2	76
N15_2	100	80	3	45
N15_3	100	150	3	66
N20_1	300	100	8	54
N20_2	300	100	8	64
N30_1	100	100	25	186

Table III shows the C2 algorithm population size, global iteration and iteration values which were used to obtain the best results. The population size and the depth of the search (number of iterations) are critical for the algorithm behavior and results, so it is very important to find a good balance between these parameters. The global iterations number is not that important as it is only used as a stopping criterion for the algorithm, in the beginning we exaggerated this parameter in order not to skip a good solution and understand the algorithm behavior better. It is worth noting that the population size and iterations parameters values may equal 100 for all problem instances, and the algorithm in most cases provides its best results, but the time may differ, that is why the individual instance parameters are presented.

Figure 2 presents the convergence speed graph for the largest instance N30_1. The algorithm behavior is similar for all instances. For small instances (with 10 and 15 batches) the algorithm may converge and find the best possible solution very quickly. When the instance size is growing, the algorithm converges fast (for time period t_1) to the solution which is 20-30% far from the optimum, and then it works for time periods $t_1 * 5$ to $t_1 * 20$ to obtain the best solution. This leads to an idea of the hybrid algorithm, which is explained in section 5.

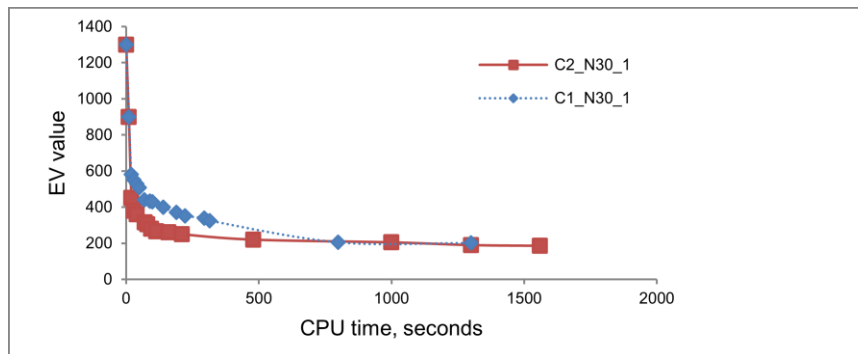


Figure 2. Convergence speed graph, C2 and C1 algorithms, instance N30_1

5 Conclusions and future work

The main conclusion is that the proposed genetic algorithm is able to efficiently (comparing to optimal solutions and another heuristic technique) solve the batch scheduling problem represented as a combination of two assignment vectors and one permutation vector and, at the same time, having one objective function. Computational results proved that the stochastic

character of the genetic algorithm is significant (random choice of genetic operators), and provides better results comparing with the deterministic operators choice.

Different solution representations and approaches demonstrated that the way of the search space definition is important for GA and by changing the solution representation the results of better quality may be obtained.

In overall, this case study demonstrated that the batch production scheduling problem, which is difficult to solve in the standard mixed-integer linear programming formulation, can be converted into the linear-programming formulation by finding the best values of the integer parameters with the help of an efficient meta-heuristic, such as proposed genetic algorithm. Also this paper described how the genetic algorithm can be efficiently used for the non-standard-GA solution representation.

Finally, we propose three statements for the future research.

- Employ parallel algorithm streams.
Taking into account the growing PC performance (especially the number of processor cores), it is reasonable considering parallel streams for the GA in order to exchange the solutions from the different streams as well as use the processor cores more efficient. One stream uses only one core that is why several parallel streams may result in the computation time profit.
- Employ the local search technique in addition to the GA.
As it is shown on the graph, the GA can quickly converge to a good solution, but it takes a while to obtain the best one. It might be reasonable to use the local search technique to explore the neighborhood of the quickly found solution and, consequently, find the next local or even the global optimum faster than by using the pure GA.
- Add more genetic operators to enhance the exploration and exploitation abilities of the algorithm.

References

1. Davis, L., Job-shop Scheduling with Genetic Algorithms, Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 136-140 (1985)
2. Engell, S., Märkert, A., Sand, G., and Schultz, R., Aggregated Scheduling of a Multiproduct Batch Plant by Two-Stage Stochastic Integer Programming, Optimization and Engineering, 5(3), pp. 335-359 (2004)
3. Goldberg, D., Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley (1989)
4. Goldberg, D. E. and Deb, K. (1991) A Comparative Analysis of Selection Schemes Used in Genetic Algorithms, Foundations of Genetic Algorithms. San Mateo, California, USA: Morgan Kaufmann Publishers, pp. 69-93.
5. Hege, L., Gicquel, C., and Minoux, M., MILP models and solution approaches for scheduling a chemical batch process, Proceedings of 24th European Conference on Operational Research, p. 297(2010).
6. Hege, L., Gicquel, C., and Minoux, M., Mathematical models and solution approaches for a chemical batch scheduling problem, unpublished, 27 pages (2010)
7. Kimms, A., A genetic algorithm for multi-level, multi-machine lot sizing and scheduling, Computers & Operations Research, 26(8), pp. 829-848 (1999)
8. Mendez, C., Cerda, J., Grossmann, I., Harjunkoski, I., and Fahl, M., State-of-the-art review of optimization methods for short-term scheduling of batch processes, Computers & Chemical Engineering, 30(6-7), pp. 913-946 (2006)
9. Moraglio, A., Towards a Geometric Unification of Evolutionary Algorithms, PhD Thesis, Department of Computer Science, University of Essex, 392 p. (2007)

10. Pinedo M., Scheduling: Theory, Algorithms and Systems, Springer: Third Edition. 671 p. (2008)
11. Ribas, I., Leisten, R., and Framiñan, J. M., Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective, *Computers & Operations Research*, 37(8), pp. 1439-1454 (2010)
12. Ruiz, R., Vázquez-Rodríguez, J. A., The hybrid flow shop scheduling problem, *European Journal of Operational Research*, 205(1), pp. 1-18. (2010)
13. Sand, G., Till, J., Tometzki, T., Urselmann, M., Engell, S., and Emmerich, M., Engineered versus standard evolutionary algorithms: A case study in batch scheduling with recourse, *Computers & Chemical Engineering*, 32(11), pp. 2706-2722 (2008)
14. Sand, G., Modeling and solving real-time scheduling problems by stochastic integer programming, *Computers & Chemical Engineering*, 28(6-7), pp. 1087-1103 (2004)
15. Wang, L., Zhang, L., and Zheng, D., An effective hybrid genetic algorithm for flow shop scheduling with limited buffers, *Computers & Operations Research*, 33(10), pp. 2960-2971 (2006)
16. Xie, J., Heuristic genetic algorithms for general capacitated lot-sizing problems, *Computers & Mathematics with Applications*, 44(1-2), pp. 263-276 (2002)

Abstracts

Schedule Recovery Strategies in Public and Freight Transportation Systems under Disruptions

Pitu Mirchandani • Jing-Quan Li

1 Introduction

Public and freight transportation systems are susceptible to unexpected costs due to unforeseen events. Minor events may not impact original service plans, while serious unexpected events definitely influence pre-determined schedules and require a recovery plan. Typical examples of such serious events include long delays, traffic accidents, vehicle breakdowns, airport closure, rail track maintenance, poor weather, road work, etc. All modes of public and freight transportation suffer from serious unexpected disruptions, including buses, trains, trucks, and airplanes. In such situations, an a priori solution may be deteriorated to such an extent that the fleet plan and attendant schedule may need to be adjusted in real-time as a function of the dynamic system state.

The following strategies can be used to recover from unexpected disruptions.

2 Modeling strategies

2.1 Adjust service starting times

Most public transportation systems provide static schedules. For example, the services of buses, trains, and airplanes provide static timetables giving starting times and ending times to passengers. Some freight transportation systems provide a time window, within which the service can start. After an unexpected disruption happens, adjusting starting time or violating the time window of related services is an approach to alleviate the disruption. Two issues need to be considered: (i) how should delays be defined? and (ii) which services should be adjusted?

2.2 Service cancellation

If a disruption is very serious, simply adjusting the starting time may not work. Some services have to be cancelled. For example, the vehicle breakdown may not only delay the current trip that is directly affected by the disruption but also other trips that the breakdown vehicle has to cover in the network. As a result, some of the delayed trips may have to be cancelled. If an

aircraft has a serious malfunction and needs to be maintained right away, some flights may have to be cancelled if there is no backup airplane. Issues include: how cancellation costs are to be computed and how cancellations and new schedules are to be determined?

2.3 Vehicle or driver reassignment

Reassigning vehicles to different services is another option to mitigate impacts of disruptions. For example, if a bus involved in an accident cannot reach the starting station of the next trip assigned to it, another bus that is near the starting station can be rescheduled to serve this trip. However, this rescheduled bus is expected to also serve the trips that are originally assigned to it. Other buses may need to be rescheduled to serve these trips. This problem turns out to be a network optimization problem. In addition, the new schedule might considerably differ from the original one. These changes may make the crew rescheduling problem difficult, since it is essential to guarantee that all drivers know the itinerary of each new trip. Thus, it is important to keep the changes in the initial schedule small in many applications.

2.4 Crew and passenger/goods reassignment

Operational problems in transportation systems are often strongly correlated. For example, the crew scheduling problem is related to the vehicle scheduling problem, for example assigning bus drivers to buses and pilots to flights. If the original plan of a vehicle is changed, the corresponding crew plan has also to be changed. The main issue here is to satisfy the worker assignment rules, such as the maximum working hours without a break. This needs to be done quickly, in near real-time, and the fact that there are already nominal assignments and schedules that only need to be perturbed makes the problem computationally feasible.

The schedule recovery problem is often a network optimization problem that considers the resources and requirements in the overall system. Following modeling and solution approaches are often used to solve the problems.

3 Modeling and solution approach

As in many operational problems, there exist two types of approaches to model the problem: a hard constraint based approach or a weighted objective based approach. The hard constraint based approach is to include a constraint for each potential issue, for example, a constraint such as “the number of new trips that the vehicle needs to serve is less than or equal to a pre-specified number”. Another example: “The total number of cancelled trips is no more than a given upper bound”. Such a modeling approach is easier to implement. However, it is difficult to determine an appropriate upper bound in real-time. If an upper bound is too large, it is actually not functional. On the other hand, if the upper bound is too small, the problem may become infeasible. Another approach is to give each objective a different weight, and the overall objective to minimize the weighted costs. For example, we can assign different weights to delays, service cancellation, schedule changes, and crew changes. But the issue then becomes how to determine the weights to assign. The presentation studies this issue.

The recovery problem is often a real-time operational problem, which needs to be solved quickly. The requirement of computational time depends on specific applications, for example, one-to-two minutes for bus rescheduling. Nevertheless, the solution algorithm has to be very fast, considering the scale of the problem. If the resulting model is simply a linear program or quadratic programming problem, a quick solution can be expected. However, if the resulting problem is an integer program, as in many cases, developing a heuristic algorithm may be the only option. Different heuristic techniques can be used, such as Lagrangian relaxation, Tabu search, large scale network search, approximate dynamic programming, etc. All the algorithms need to take the advantage of special structures of the application in order to have good performance and reduction in the computational time. The presentation will discuss and compare some solution approaches.

Train Scheduling in the UK using a Job Shop Scheduling Approach

Banafsheh Khosravi • Julia A. Bennell • Chris N. Potts

1 Introduction

Today, the railway network system is a major part of the transportation infrastructure in many countries. Due to the increasing volume of railway traffic and the high expense of construction or modification of infrastructure, train scheduling is employed to ensure the best usage of the existing capacity. There has been increasing interest in the train scheduling problem recently and job shop scheduling has been exploited in some studies after the pioneering publication of Szpigel [13]. Later, Sahin [12], Oliveira and Smith [8], Rodriguez [10], D'Ariano et al. [4], Corman et al. [3], and Liu and Kozan [6] have also formulated the problem as a job shop scheduling problem. Whereas Szpigel [12] and Sahin [11] use mathematical programming techniques for the job shop scheduling problem, Oliveira and Smith [8] and Rodriguez [9] offer constraint programming approaches. Moreover, D'Ariano et al. [4], Corman et al. [3], and Liu and Kozan [6] suggest an alternative graph formulation of the job shop scheduling problem.

This study aims to achieve further developments including refinements to existing models to consider more realistic and detailed constraints in railway operations although many studies are still a crude approximation of reality. Moreover, we develop solution methods inspired by the successful results of the shifting bottleneck procedure and local search techniques to solve job shop scheduling problems. The proposed methods are implemented on a real-world case study based on London and South East area of the UK that is a dense and complex network of interconnected lines for commuters from/to London, East Sussex, and the Channel Tunnel, and for freight from/to the Kent coast.

Banafsheh Khosravi

School of Management / CORMSIS, University of Southampton, SO17 1BJ, UK

E-mail: b.khosravi@soton.ac.uk

Julia A. Bennell

School of Management / CORMSIS, University of Southampton, SO17 1BJ, UK

E-mail: j.a.bennell@soton.ac.uk

Chris N. Potts

School of Mathematics / CORMSIS, University of Southampton, SO17 1BJ, UK

E-mail: c.n.potts@soton.ac.uk

2 Problem definition

Researchers approach the problem as either a microscopic or macroscopic scheduling problem depending on the level of the information and details on the track topology and train dynamics considered [2]. This paper addresses the train scheduling problem at the micro level including detailed information about the tracks and train movements. We base our experiments on a bottleneck area in the South East of the UK where the network has a complicated structure including junctions and stations. Whereas many earlier researchers investigate networks with only passenger or freight traffic, our network has a mixed traffic.

The aim is to define entry times and the order of trains on track sections for predetermined routes to go from origin to destination. In the case of disruptions, which cannot be avoided in an operational environment, the same scheduling technique can be applied for real-time traffic management when the solution method is fast enough. The resulting schedule must be conflict-free, where a conflict is defined as two or more trains assigned to the same track section. It is also important to consider the possibility of delay propagation in a rail network, which results from the high interdependency of the trains. Thus, the objective is to minimize the total weighted tardiness to avoid delay propagation. There are other objective functions common in the train scheduling studies such as minimizing operational cost, minimizing transit time which is the total time for a train to complete its itineraries, minimizing deviation from some ideal or preferred timetable, and maximizing passenger satisfaction by maximizing direct connections or minimizing passengers' waiting time. There are obviously different operational and safety constraints to be taken into account for train scheduling problem.

3 Problem Formulation

There is a strong similarity between train scheduling problems and the well-known job shop scheduling problem. Job shop scheduling allocates machines to competing jobs in a way that a machine can process only one job at a time. Also, line blocking is a safety principle for train movement that divides the network into sections called blocks which can be occupied by only one train at a time. Hence, a train traversing a block is analogous to a job being processed on a machine. The job shop problem is modelled first as a Mixed Integer Linear Programming (MILP) to minimize the total weighted tardiness of the jobs. Tardiness is calculated based on the due date of the job which is equivalent to the pre-defined time that a train should leave the partial network. The release time of each job is defined as the entering time of each train to the partial network. Weights can be determined regarding the train priorities.

Modelling a job shop scheduling problem with a disjunctive graph does not take into account the buffer capacity between consecutive machines, which seems to be an important issue in many real-life scheduling problems. As a modification of the disjunctive graph, the alternative graph by Mascis and Pacciarelli [7] addresses this restriction and it can be adapted to formulate different types of constraints. We also use a modified version of the disjunctive graph to represent the train scheduling problem. The model includes several types of constraints. Running time and dwell time constraints correspond to disjunctive and conjunctive constraints in the disjunctive graph formulation [11]. Headway constraints for trains should be added to consider the minimum interval time between two consecutive trains. Another set of constraints is added to consider a blocking job shop problem which forces jobs to remain on a machine after their completion until the next machine becomes available.

4 Solution Method

The train scheduling problem is a very large and complex combinatorial problem which is known to be NP-complete (see [14] and [5]). Thus, it seems to be more appropriate to use heuristic methods including local search procedures, which produce the most promising results

for the job shop scheduling problems. The shifting bottleneck procedure of Adams et al. [1] is known to be one of the most fruitful methods for solving the job shop scheduling problem considering the disjunctive graph formulation. A modified heuristic method based on the shifting bottleneck procedure is proposed for train scheduling problem. This new approach differs from the conventional approach in finding the bottleneck and solving the single machine problem. It adapts Pinedo and Singer's method [9] for train scheduling to include additional constraints. Moreover, a local search algorithm is presented for the considered train scheduling problem.

The code is developed in MS Visual C++ 2010. Computational experiments provide a comparative assessment of the employed methods against the FCFS dispatching rule in terms of their solution quality and computational time. Results are reported for an area of about 15 km long with totally 136 unidirectional and bidirectional blocks. We consider 27 passenger trains in off-peak services in a cycle length of 30 minutes, and freight trains that are acyclic.

Acknowledgements We thank School of Management, previous LASS Faculty of the University of Southampton and the LANCS Initiative for partially funding and supporting this project. We are also grateful to Southeastern, the train operating company, for providing data.

References

1. Adams J., Balas E., Zawack D., The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34, 391-401 (1988)
2. Caimi G., Algorithmic decision support for train scheduling in a large and highly utilised railway network, PhD thesis, Swiss Federal Institute of Technology Zurich (2009).
3. Corman F., D'Ariano A., Pacciarelli D., Pranzo M., A tabu search algorithm for rerouting trains during rail operations, *Transportation Research Part B*, 44, 175-192 (2010)
4. D'Ariano A., Pacciarelli D., Pranzo M., A branch-and-bound algorithm for scheduling trains in a railway network, *European Journal of Operational Research*, 183, 643-657 (2007)
5. Garey M. R., Johnson D. S., *Computers and Intractability: A Guide to Theory of NP-Completeness*, New York: Freeman (1979)
6. Liu S.Q., Kozan E., Scheduling trains as a blocking parallel-machine shop scheduling problem, *Computers & Operations Research*, 36, 2840-2852 (2009)
7. Mascis A., Pacciarelli D., Job shop scheduling with blocking and no-wait constraints, *European Journal of Operational Research*, 143, 498-517 (2002)
8. Oliveira E., Smith, B. M., A Job-Shop Scheduling Model for the Single-Track Railway Scheduling Problem, Tech. Rep. No. 21, England: School of Computing, University of Leeds (2000)
9. Pinedo M., Singer M. (1998). A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics* 46: 1-17.
10. Rodriguez J., A constraint programming model for real-time trains scheduling at junctions, *Transportation Research Part B*, 41, 231-245 (2007)
11. Roy B., Sussman R., Les problemes d'ordonnancement avec contraintes disjonctives, Tech. Rep. No. 9, SEMA, Paris (1964)
12. Sahin I., Railway traffic control and train scheduling based on inter-train conflict management, *Transportation Research, Part B*, 33, 511-534 (1999)
13. Szpigel B., Optimal Train Scheduling on a Single Track Railway, In M. Ross (Ed.), *Operational Research '72*, 343-352, Amsterdam, the Netherlands (1973)
14. Ullman J., NP-complete scheduling problems, *Journal of Computer and System Science*, 10, 384-393 (1975)

Optimization of Scheduling for Large-Scale Logistics Tasks

Rodrigo Silva-Lugo · Robert Kozma ·
Ludmilla Werbos · Giovanni E. Pazienza

1 Introduction

Efficient scheduling is a crucial issue in many areas belonging to both defense and civilian domains, including disaster response, power distribution, transportation, manufacturing job scheduling etc. [1–3]. Throughout the years, there have been countless efforts to tackle this problem; nevertheless, the appearance on the market of multi-core platforms, and advanced Field Programmable Gate Arrays (FPGAs) offer unexplored possibilities, which require novel and more efficient approaches. The ultimate goal is to develop efficient nonlinear optimizers, like approximate dynamic programming, though we are far away from this goal: today's state of art optimization tools explore mixed integer programming techniques [4].

In the present work, we use the Gurobi software to explore and understand better the problem of large scale optimization tasks, which may include hundreds of thousands of variables. In particular, we focus on the decision support to optimize the assignment of resources for the domestic market operation of a large distribution company. Our Efficient Scheduling Process (ESP) is composed of three main steps, each of them formulated as a Mixed Integer Programming (MIP) problem: 1) determine point to point

Rodrigo Silva-Lugo
Center for Large Scale Integrated Optimization & Networks (CLION)
FedEx Institute of Technology, University of Memphis
E-mail: rsilva@memphis.edu

Robert Kozma
Center for Large Scale Integrated Optimization & Networks (CLION)
FedEx Institute of Technology, University of Memphis
E-mail: rkozma@memphis.edu

Ludmilla Werbos
Center for Large Scale Integrated Optimization & Networks (CLION)
IntControl LLC
E-mail: l.dalmat@gmail.com

Giovanni E. Pazienza
Center for Large Scale Integrated Optimization & Networks (CLION)
FedEx Institute of Technology, University of Memphis
Pazmany University, Budapest
E-mail: g.pazienza@memphis.edu

flight and volume; 2) assign dispatch locations and volumes; 3) assign fleet vehicles to move pieces between dispatch locations and volumes. The ESP must accomplish the following tasks: 1) maximize the profit; 2) assign available resources while minimizing costs and respecting operational constraints; 3) find the optimal cost resource assignment to move the forecasted volume of products, adhering to operational constraints; 4) globally re-flow all forecasted volume to minimize costs according to capacity and operational constraints [5].

2 Outline of the experiments and results

The experiments were performed on a Dell PowerEdge server with 24 cores and 48 GB of RAM with a Linux Red Hat operative system. As an optimization software, we used the well-known Gurobi Optimizer. We decomposed the original problem into three subproblems, denoted by D, N, and C. The details of such decomposition are not provided in this abstract but they will be available on the full version of the paper. The first step of our experiments has been running all possible combination of parameters in a reduced version of the original problem. Next, we selected the best five sets of parameters based on the *shortest* time required to reach 1% MIP gap (STO), and the *least* number of nodes explored (LNE) during the branch and bound phase. Finally, we simulated two different versions of the optimization in subproblem C.

The Gurobi parameters changed through the experiments are: *BarOrder*, *Crossover*, *Crossover Basis*, *Cuts*, and *MIPFocus*. The values for the best results obtained with the subsystem N are shown in Tables 1 and 2; a complete description of these parameters can be found in [6]. The results for the experiments used both available versions of step 3 in subproblem C. These two versions are slightly different since they were obtained using a different forecast. Table 3 shows the results for the parameters sets obtained using the STO criteria. Finally, Table 4 show the results for the parameters corresponding to LNE, but only for version 2 of the problem, since for all five sets of parameters, version 1 never reached the desired MIP gap in less than 24 hours.

It is evident that the greatest improvement was achieved for version 2 of step 3 (C), with an average time of around 9 hours, much better than 24 hours usually required for this step. However, the quality of this result is not uniform through the other configurations: for example, for version 1 of step 3 (C), all experiments ran for 24 hours, without reaching the desired MIP gap of 1%. It is also worth mentioning that the custom hardware and software configurations may have had an effect in the performance improvement.

Table 1 MIP Parameters for Faster Runs of Step 3 (N)

	BarOrder	Crossover	Cross. Basis	Cuts	MIPFocus
Set 1	A. M. Degree	Auto	Quick	Very aggr.	Feasibility
Set 2	Auto	Auto	Quick	Very aggr.	Feasibility
Set 3	Auto	Disabled	Quick	Very aggr.	Feasibility
Set 4	A. M. Degree	Disabled	Quick	Very aggr.	Feasibility
Set 5	N. dissection	Primal	Quick	Aggressive	Feasibility

Table 2 MIP Parameters for Least Nodes Explored in Night Step 3 (N)

	BarOrder	Crossover	Cross. Basis	Cuts	MIPFocus
Set 1	Auto	D/P-P	Slow / stable	Auto	BOB
Set 2	Auto	D/P-P	Slow / stable	Conservative	BOB
Set 3	A. M. Degree	D/P-P	Slow / stable	Very aggr.	BOB
Set 4	A. M. Degree	D/P-P	Slow / stable	Auto	BOB
Set 5	N. dissection	D/P-D	Slow / stable	Very aggr.	BOB

3 Conclusions and Future Work

By selecting the optimal set of control parameters for Gurobi 3.0 out of 720 parameter sets, we were able to determine a set of parameters that significantly increased the performance of the system. Important progress was made in the understanding of the details of the data used for the generation of the MIP models by the distribution company. One of the qualitative results was finding the difference between the sets of parameters which are most appropriate for memory-intensive problems versus problems requiring higher processor speed. Running the simplified version of the problem to work through the sets of parameters faster allowed us to find structural similarities and differences between various partitionings of the original (subproblem C) problem, which in turn lead us to better understanding of the influence of the choice of Gurobi control parameters on the speed of convergence. Future directions of research will include expanding the problem both vertically (up to to several days) and horizontally (combining separate solution stages into integrated model). However, the complexity of such integrated model might approach the limit of traditional Mixed Integer Pro-

Table 3 Experiment Time and Performance for Step 3 (C) with STO Parameters (Average and standard deviation only for sets that reached 1 % MIP gap in less than 24 hours. Runs marked with * did not reach 1% MIP gap before the 24 hours limit)

	Version 1		Version 2	
	Time (h)	MIP Gap (%)	Time (h)	MIP Gap (%)
Set 1	24.0*	1.16*	5.07	0.99
Set 2	10.3	0.95	5.30	0.99
Set 3	10.6	0.95	7.91	0.98
Set 4	24.0*	1.16*	7.96	0.98
Set 5	24.0*	1.01*	18.4	1.00
Avg.	10.45	0.95	8.93	0.99
Std. Dev.	0.21	0.00	5.47	0.01

Table 4 Experiment Time and Performance for Step 3 (C) with LNE Parameters (Average and standard deviation only for sets that reached 1 % MIP gap in less than 24 hours. Runs marked with * did not reach 1% MIP gap before the 24 hours limit)

	Version 2	
	Time (h)	MIP Gap (%)
Set 1	12.92	0.99
Set 2	24.00*	1.66*
Set 3	13.20	0.99
Set 4	24.00*	1.66*
Set 5	10.78	1.00
Avg.	12.30	0.99
Std. Dev.	1.32	0.00

gramming methods, calling for alternative methods of solution, such as approximate dynamic programming problem [7], which will provide further increase in the performance necessary for the daily operation of the distribution company.

Acknowledgements This work would not have been possible without the support and cooperation of the the FedEx Institute of Technology and FedEx Express. Special thanks to John Gunckel, James Segars, Chris Holliday, Raymond Igharas, J Q Chen, and Wei He.

References

1. A. S. Ko and N.-B. Chang, "Optimal planning of co-firing alternative fuels with coal in a power plant by grey nonlinear mixed integer programming model," *Journal of Environmental Management*, vol. 88, no. 1, pp. 11 – 27, 2008.
2. L. Contesse, J. Ferrer, and S. Maturana, "A mixed-integer programming model for gas purchase and transportation," *Annals of Operations Research*, vol. 139, pp. 39–63, 2005.
3. S. Kesen, S. Das, and Z. Gungor, "A mixed integer programming formulation for scheduling of virtual manufacturing cells (VMCs)," *The International Journal of Advanced Manufacturing Technology*, vol. 47, pp. 665–678, 2010.
4. L. Werbos and P. Werbos, "Self-organization in CNN-based object nets," in *12th International Workshop on Cellular Nanoscale Networks and Their Applications (CNNA)*, 2010, pp. 1–6.
5. L. Werbos, R. Kozma, R. Silva-Lugo, G. E. Pazienza, and P. Werbos, "Metamodeling for large-scale optimization tasks based on object networks," in *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2011*, 2011.
6. "Gurobi parameters," December 2010. [Online]. Available: <http://gurobi.com/doc/40/refman/node572.html>
7. W. Powell, *Approximate dynamic programming: solving the curses of dimensionality*. John Wiley & Sons, 2007.

Two-level water-flow algorithm for vehicle routing problems

Tran Trung Hieu • Ng Kien Ming

1 Introduction

In the field of logistics and supply chain management, how to arrange an appropriate supplier-to-customer assignment and determine an efficient distribution routing is very important. These problems are broadly classified as vehicle routing problems (VRPs), which are NP-hard optimization problems.

Water-flow algorithm (WFA) is a metaheuristic that could be used to solve certain types of NP-hard optimization problems. An important application of this algorithm is in solving the flexible flow shop scheduling with limited or unlimited intermediate buffers [1]. The WFA is inspired by the hydrological cycle in meteorology and the erosion phenomenon in nature. The simulation of natural phenomena in this algorithm aims to obtain a balance between its solution exploration and exploitation capabilities. The main operators of the WFA are constructed based on properties of water flow, such as drops of water (*DOWs*) always moving to lower positions, distributing to many regions on the ground, and eroding terrain.

In this paper, we propose a two-level WFA (2LWFA) for solving VRPs. The first level of the proposed algorithm is to solve the mathematical programming model of the VRPs with the relaxation of the integrality constraints. At the second level, a modified WFA is then applied to search for improved or optimal solutions from the initial solutions obtained from the first level. Here, we illustrate the performance of the 2LWFA with the capacitated vehicle routing problem (CVRP).

2 Capacitated vehicle routing problem

The CVRP can be considered as one of the typical VRPs. This problem consists of a fleet of vehicles with pre-specified limited capacity, which has to serve a set of customers with specific demands. In our work, we have focused on the CVRP with one depot.

Tran Trung Hieu

Department of Industrial and Systems Engineering, National University of Singapore,
10 Kent Ridge Crescent, Singapore 119260
E-mail: g0600324@nus.edu.sg

Ng Kien Ming

Department of Industrial and Systems Engineering, National University of Singapore,
10 Kent Ridge Crescent, Singapore 119260
E-mail: kienming@nus.edu.sg

There are many research works related to the CVRP. Among them, the works that formulate the CVRP as an integer linear programming problem have generally led to effective exact solution approaches for the problem. In this paper, we have used the integer linear programming formulation presented in [2] and corrected in [3].

3 Two-level WFA for VRPs

3.1 Constructing initial solutions (first level)

While it is possible to apply the WFA to solve the CVRP directly, the CVRP model could be helpful in finding good initial solutions and reducing the computation effort when applying the WFA. However, the efficiency of solving the CVRP model depends on the solver used. Most solvers often require large amounts of computation time and may fail to return solutions for problems with large size, especially for integer programming models. Thus, we focus on solving a linear programming relaxation of the CVRP model.

In [3], X_{ij} is a binary variable that is equal to 1 if and only if customers (or depot) i and j are connected. We relax these variables to real numbers in $[0, 1]$ and solve the resulting relaxed model (with all the other constraints in [3] unchanged) using the commercial optimization package LINGO 5.0. For the optimal solution that is obtained this way, we arrange the resulting values of decision variables in a descending order. The X_{ij} variables with large values can be considered to be potential connections of customers (or depot) i and j . As such, we will assign these X_{ij} variables to have a value of 1. Also, we check the feasibility of the assignment based on the constraints of the CVRP and fit the most appropriate assignment. This check-and-fit procedure is performed iteratively until a feasible solution of the CVRP is determined. In this paper, we illustrate a simple example of this procedure carried out on a benchmark instance of Christofides and Eilon [4] in Section 4. Next, a perturbation scheme based on the 2-opt algorithm is used to generate a set of seed solutions from the initial feasible solution.

3.2 Modified WFA (second level)

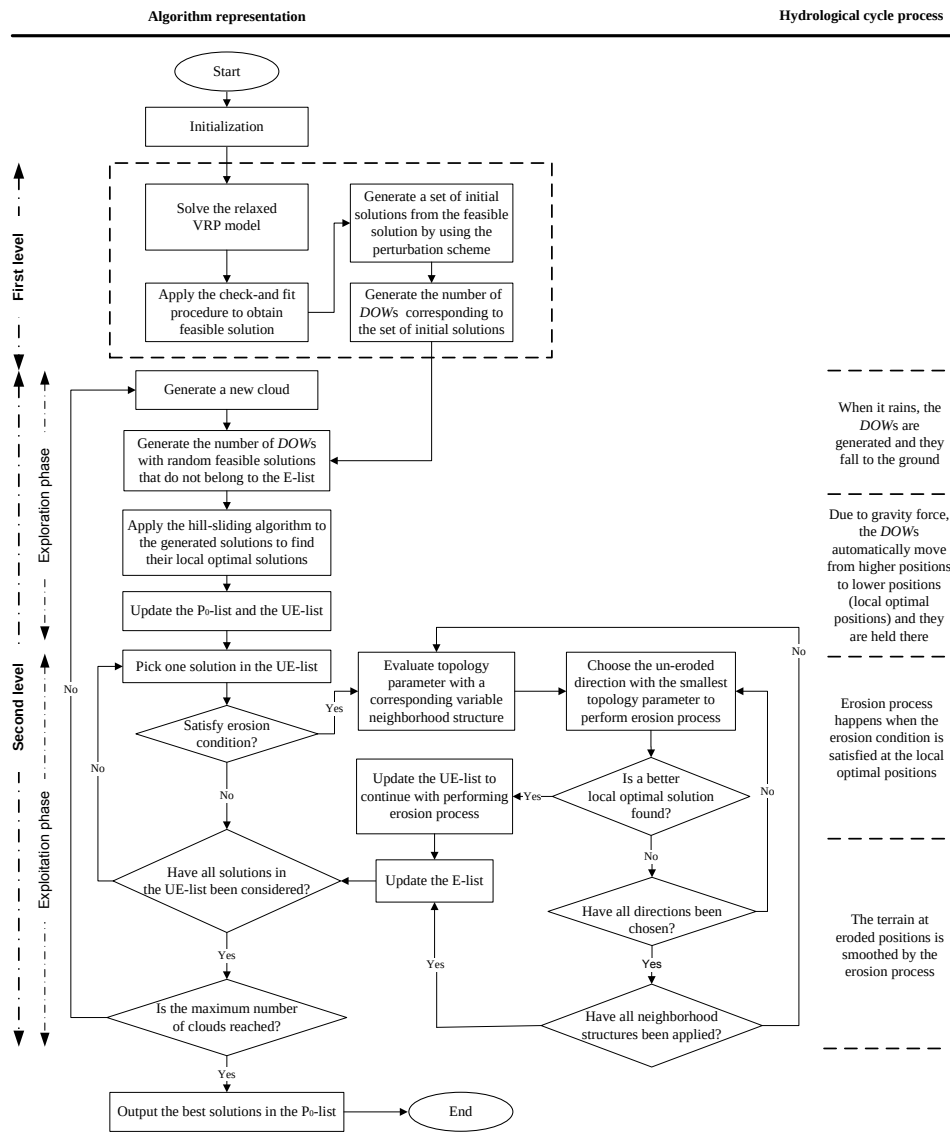
After constructing a set of seed solutions, a modified WFA is applied to search for improved solutions from the seed solutions. In particular, we modify the WFA in [1] by using a new solution representation for the CVRP. We also set the erosion capability of $DOWs$ in the modified WFA to be a constant, which helps to simplify the calculations involved in the erosion process. Moreover, a variable neighborhood structure based on the k -opt algorithm (with $k = 3, 4$) is used in the erosion process to enhance the flexibility and efficiency of this process.

A flow chart summarizing the 2LWFA for VRPs is shown in Figure 1.

4 An illustrative example

A CVRP benchmark instance of Christofides and Eilon drawn from [4] is used to illustrate how to determine a feasible solution from the optimal solution obtained by solving the relaxed mathematical programming model. This instance, known as E-n13-k4, consists of 12 customers served by 4 vehicles. It is solved by LINGO 5.0 and we obtain the values of decision variables as follows: $X_{1,2} = X_{1,3} = X_{1,7} = X_{1,10} = X_{2,1} = X_{3,1} = X_{4,1} = X_{7,1} = X_{10,13} = X_{13,4} = 1$; $X_{5,8} = X_{5,12} = X_{6,9} = X_{6,11} = X_{8,5} = X_{8,11} = X_{9,6} = X_{9,12} = X_{11,6} = X_{11,8} = X_{12,5} = X_{12,9} = 0.5$; $X_{3,8} = 0.28$; and all the rest of the variables $X_{ij} = 0$ (here, index 1 denotes the depot). After applying the check-and-fit procedure, we construct a feasible solution with 4 routes, i.e., route 1: (2, 9), route 2: (3, 8, 5, 12), route 3: (7, 11, 6), and route 4: (10, 13, 4). The total corresponding travel distance is 307.

The feasible solution is used to generate a set of seed solutions that are then used as initial solutions in the modified WFA. The final solution obtained by this algorithm has total travel distance 247, which is the same as the best known solution from the literature.



5 Preliminary experiments

The 2LWFA has been coded using Visual Basic 6.0 linked with LINGO 5.0, and preliminary experiments have been performed on an Intel Centrino Duo 1.60 GHz CPU with 1.5 GB of RAM. These experiments are carried out on the CVRP instances of Christofides and Eilon taken from [4] and shown in Table 1.

Table 1. Preliminary experiment results

Instance	Optimal/Best known value	Initial solution after applying check-and-fit procedure	2LWFA	CPU time (secs)	
				Lingo	WFA
E-n13-k4	247	307	247	0.2	0.50
E-n22-k4	375	390	375	0.5	1.75
E-n23-k3	569	623	569	0.6	4.47
E-n30-k3	534	582	534	1.0	7.50
E-n33-k4	835	910	835	1.5	35.25
E-n51-k5	521	627	521	3.0	120.15
E-n76-k7	683	843	687	18.0	1261.72
E-n101-k8	817	1012	853	50.0	4840.14

From Table 1, we see that the 2LWFA can obtain optimal solutions for the instances with small and medium size within reasonable computation time. For instances with larger size, i.e., E-n76-k7 and E-n101-k8, the proposed algorithm can also find solutions with a relative deviation of 0.6% and 4.4% over the optimal/best known solution, respectively.

In addition, the results from Table 1 show that the solution obtained by 2LWFA achieved an average improvement of 12.46% over the initial solution obtained by applying the check-and-fit procedure to the solution found using LINGO. We also find that these improvements are more significant for the instances with larger size.

6 Conclusions

The experimental results show that the 2LWFA can solve the CVRP efficiently. We have continued to carry out experiments based on other benchmark CVRP instances drawn from the literature. The results also show the potential of the 2LWFA to solve other types or variants of VRPs.

References

1. Tran T.H. and Ng K.M. *A Water Flow Algorithm for Flexible Flow Shop Scheduling with Intermediate Buffers*. Journal of Scheduling, in press (2010).
2. Kulkarni R.V. and Bhavé P.R. *Integer Programming Formulations of Vehicle Routing Problems*. European Journal of Operational Research, 20, 58–67 (1985).
3. Imdat K., Gilbert L. and Tolga B. *A Note on the Lifted Miller–Tucker–Zemlin Tour Elimination Constraints for the Capacitated Vehicle Routing Problem*. European Journal of Operational Research, 158, 793–795 (2004).
4. <http://neo.lcc.uma.es/radi-aeb/WebVRP/>

Supply Logistics Synchronization

Scale Development and Empirical Study of Chinese Manufacturing

Lei Xie • Shihua Ma • Pei Huang • Huaming Gui

Abstract In assembly supply chain, because manufacturers demand a variety of related components, the supply logistics management is difficult. A two-dimensional supply logistics synchronization theory is proposed, based on enterprise investigation, literature review and experts interview. It considers not only the synchronization between multiple suppliers and downstream manufacturer, but also the various suppliers collaborate with each other to meet the needs of downstream manufacturer. The scale to measure supply logistics synchronization is developed, it includes 3 dimensions: suppliers and manufacturer synchronization, synchronization among suppliers and logistics service capability. We collect data by questionnaire from 191 Chinese organizations to test and validate the scale. The scale has good reliability and validity; scholars could use it in supply chain research, and it provides standards for companies to examine their supply chain.

Key words: supply chain; supply logistics synchronization; scale development; empirical study

1 Introduction

In the upstream of assembly supply chain, varieties, huge numbers of components from the different suppliers join together at the manufacturing facility. Such a supply chain clearly requires that all components should be presented when completing the product. Various suppliers may be located in different place, have different process, and only seek to maximize their own interests, so the supply chain usually disrupts because of one or two components shortages. Therefore, it could significantly improve the responsiveness and business performance of the supply chain by making the components from different vendors arriving manufacturer simultaneously and synchronizing the supply logistics of components.

Supply logistics synchronization have been concerned by researchers gradually. Simatupang and Sridharan (2005) propose an instrument to measure the extent of collaboration in a supply chain [1], Sanders (2007), Singh and Power (2009) use structural equation modeling to empirical research supply chain coordination, which are related to the supply logistics synchronization [2][3]. But existing researches are not enough to fully reflect the complexity of supply stage, and this important part of the supply chain is lack of empirical researches especially. This paper extends the theory of supply logistics synchronization by developing a scale to measure it. Data for the study are collected by questionnaire and the measurement scale is tested and validated using structural equation modeling. It is hoped that this study will find the main factors of supply logistics synchronization, deepen the researchers' understanding of the issues of supply logistics, lay the foundation for subsequent researches and provide real, actionable recommendations for enterprises.

2 Supply logistics synchronization

We introduce a general-purpose definition of supply logistics synchronization as follows:

Supply logistics synchronization is based on resource integration in the supply chain, all components supplied simultaneously and matched with the product assembly by various vendors in supply chain, thus reducing the delayed delivery and rising inventory costs which caused by components shortages. So that it could improve the responsiveness, timely and accurate manner to meet customer needs, and the competitive ability of the entire supply chain.

We stand in the position of supplier to measure supply logistics synchronization from three dimensions: suppliers and manufacturer synchronization, synchronization among suppliers and logistics service capability.

Lei Xie

School of Management, Huazhong University of Science and Technology, Wuhan, China

E-mail: lei_xml@hotmail.com

Shihua Ma

School of Management, Huazhong University of Science and Technology, Wuhan, China

E-mail: shihuama@mail.hust.edu.cn

Pei Huang

E-Works Company, Wuhan, China

Huaming Gui

School of Business, Heibei University, Wuhan, China

2.1 Suppliers and manufacturer synchronization

Suppliers and manufacturer synchronization is based on a series of supply and demand in the supply chain, taking manufacturer as the custom of various suppliers, and then building long-term partnership, so that they could cooperate fully in key business processes by means of information sharing and system integration. The related literature of this dimension: Sanders (2007) [2]; Singh, Power (2009) [3]; Li et al (2005) [4]; Braunscheidel, Suresh (2009) [5]; Li et al (2006) [6]; Huang et al (2008) [7].

2.2 Synchronization among suppliers

Manufacturers need a variety of components for assembly, and the needs of components are closely related with each other. To achieve supply logistics synchronization needs not only synchronization between suppliers and manufacturer, but also the various suppliers synchronizing with each other. Synchronization among suppliers is not just considering the relationship between the two suppliers, but numbers of suppliers cooperate with each other to meet the components demand of their mutual customer —downstream manufacturer. The related literature of this dimension: Lin(2006) [8]; Modi, Mabert (2007) [9]; Mondragon , Lyons (2008) [10]; Cousins(2005) [11].

2.3 Logistics service capability

Supply logistics synchronization is affected by logistics service capability. Logistics service capability comprises not only the processing power of logistics facilities and resources, such as warehousing, transportation, sorting, more importantly, planning, organization and the control capability of logistics activities. Logistics services may be carried out by the suppliers, manufacturer or third-party service providers. The related literature of this dimension: Zimmer (2002) [12]; Panayides, So(2005)[13]; Kaynak, Hartley(2006)[14].

3 Scale development

Our research team investigated a number of assembly supply chains, understood the actual situation of organizations. Based on these information and previous literature, the items for supply logistics synchronization were generated.

Conceptual construct validation was carried out following Q-sort of Moore and Benbasat (1991) [15]. Four rounds of sorting the items were carried out, each involving two judges who were academicians in supply chain management. Each judge participated only in one round. In Round 1, the judges sorted the items into independent constructs. In Round 2, each judge sorted items according to given constructs. Based on these results, ambiguous questions were reworded. Rounds 3 and 4 were repeats of Rounds 1 and 2 respectively, to ensure stability and clarity of questions.

Two evaluation indices are used to measure inter-judge agreement level: Hit Ratio and Cohen's Kappa. Hit Ratio analyzes how many items are placed by the panel of judges for each round within the target construct. Cohen's Kappa is a measure of agreement that can be interpreted as the proportion of joint judgment in which there is agreement after chance agreement is excluded. The recommended value of Hit Ratio is 0.7, for Kappa, scores greater than 0.65 are acceptable [15] [16]. Table 1 presents a summary of agreement scores for the four rounds.

Table 1 *Inter-judge agreements*

Agreement measure	Round 1	Round 2	Round 3	Round 4
Hit Ratio	0.87	0.89	0.91	0.98
Cohen's Kappa	0.62	0.79	0.73	0.93

Then these items were reviewed by 3 professor and associate professor, 2 managers in automotive and electronics industries. Based on the feedback from them, redundant and ambiguous items were either modified or eliminated, 17 items with 3 dimensions were retained. Measurement items for supply logistics synchronization and related literature are listed in Table 3.

4 Research methods

4.1 Data collection

This study used survey methods to collect data. We cooperated with E-Works to distributed questionnaires. The questionnaires were completed by mail and online forms, using a 5-point Likert scale (1—low, 5—high). E-Works is an online media, vertical portal, service and research organization focused on IT application of Chinese Manufacturing. E-Works portal (www.e-works.net.cn in Chinese and www.eworksglobal.com in English) is the most popular and authoritative portal of Chinese manufacturing IT application area.

By the end of December 15, 2010, we collected data from 191 organizations, 11.6% “Auto Industry”, 22.1% “Electronic and Other Electric Equipment”, 20.2% “Machinery Equipment”, 25.3% “Raw Materials Products”, including manufactures in the areas of household appliances, clothing and food processing, and so on. The data could reflect the status of manufacturing in China. 48.4% of the respondents were senior managers, the middle managers were 37.1%, the consultants and others were 14.5%. The respondents are expected to have a broad view of supply chain practices in their organization.

4.2 Statistical analysis

We conduct confirmatory factor analysis (CFA) by using AMOS7.0 and SPSS16.0. The measurement model is showed in Figure 1, items of suppliers and manufacturer synchronization are showed as SM1 to SM8, items of synchronization among suppliers are showed as SS1 to SS5 and items of logistics service capability are showed as LS1 to LS4.

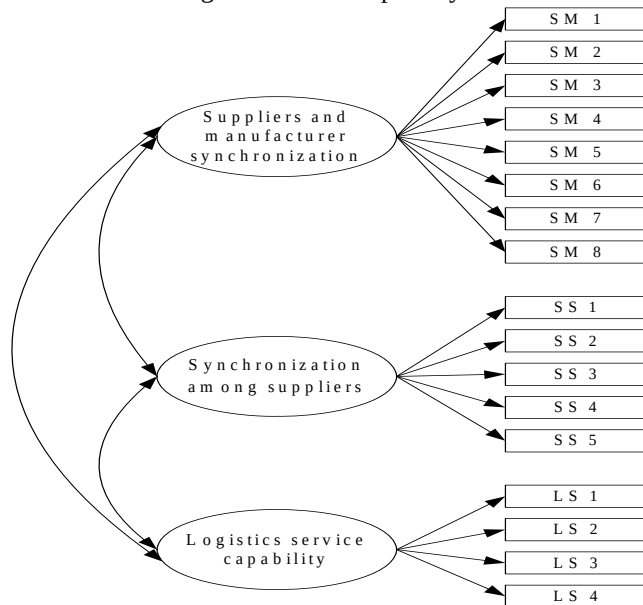


Figure 1. Measurement Model

The first step is to assess overall model fit. In Table 2, we provide goodness-of-fit criteria for the measurement model. A model is considered to be satisfactory if $1 < \chi^2 / df < 3$, GFI, NFI, IFI, TLI, CFI > 0.85 , and RMR, RMSEA < 0.08 . Results indicate that the measurement model has acceptable fit.

Table 2 Model fit statistics

χ^2 / df	GFI	NFI	IFI	TLI	CFI	RMR	RMSEA
2.191	0.869	0.880	0.931	0.914	0.930	0.073	0.079

The results of CFA are showed in Table 3. Factor loading size and significance are provided to assess validity. As Fornell and Larcker (1981), it is a strong evidence for

wonderful validity that the squared factor loading is greater than 0.5, and validity can be acceptable if that Standardized factor loading is greater than 0.5[17]. As shown in Table 3, standardized factor loadings for all measures are greater than 0.5. In addition, the average variance extracted (AVE) of synchronization of suppliers and manufacturer is slightly less than 0.5, and the others AVEs are greatly above 0.5, this also shows that the scale has good validity.

The reliability of the instrument is assessed by evaluating Cronbach's α and Composite reliability (CR) of the dimension measures. A scale is said to be reliable if Cronbach's $\alpha > 0.7$ and $CR > 0.7$. As shown in Table 3, Cronbach's α and CR for the dimension measure are all quite higher than 0.7.

Table 3 *Validity and reliability evaluation*

Factor	Item	Standardized factor loading	α	CR	AVE
Suppliers and manufacturer synchronization	Joint demand forecasting	.639	.879	.881	.483
	Collaborative planning	.591			
	Frequently interacting with customers to set standards	.697			
	Frequently measuring and evaluating customer satisfaction	.683			
	Frequently determining customer expectations	.751			
	Facilitating customer to seek assistance	.784			
	Periodically evaluating the relationship with customers	.753			
	Customer feedback	.638			
Synchronization among suppliers	Joint processing orders	.765	.882	.880	.597
	Joint R&D	.840			
	Joint confirming delivery quantities	.844			
	Sharing production plans	.723			
	Marching all components designated for assembly as a kit	.679			
Logistics service capability	JIT delivery	.713	.882	.887	.664
	Timely responding to delivery demand	.827			
	Sharing information during the distribution	.863			
	Marching schedule of transportation and assembly	.847			

5 Discussion

This study provides a validated instrument for measuring supply logistics synchronization. The development process included literature review, expert interviews, Q-sort and large sample survey. The result is a scale of 3 dimensions: synchronization of suppliers and manufacturer, synchronization among suppliers, logistics service capability, and 17 items. This instrument is validated, providing a high degree of confidence in the scales' validity and reliability. The scale can be used by scholars in supply chain research, and provides standards for companies to examine their supply chain.

Some limitations should be noted. To establish evidence of statistical generalizability, we need to repeat the study with more samples. We should apply this scale to supply chain practice researches, thus draw more conclusions beyond the scope of our sample. Finding out

how supply logistics synchronization affects supply chain agility, performance, etc, which will serve as the next step of the research.

Acknowledgements The authors wish to thank the experts and researchers of E-Works Company. This research was supported by the National Natural Science Foundation of China (No.71072035; No.71071050).

References

1. Simatupang T.M., Sridharan R. The collaboration index: a measure for supply chain collaboration. *International Journal of Physical Distribution & Logistics Management*, 35(1): 44-62 (2005).
2. Sanders N. R. An empirical study of the impact of e-business technologies on organizational collaboration and performance. *Journal of Operations Management*, 25(6): 1332-1347(2007).
3. Singh P. J., Power D. The nature and effectiveness of collaboration between firms, their customers and suppliers: a supply chain perspective. *Supply Chain Management: An International Journal*, 14(3): 189–200 (2009).
4. Li S.H., Raob S.S., Ragu-Nathanb T.S., Ragu-Nathanb B.. Development and validation of a measurement instrument for studying supply chain management practices. *Journal of Operations Management*, 23(6): 618-641(2005).
5. Braunscheidel M.J., Suresh N.C. The organizational antecedents of a firm's supply chain agility for risk mitigation and response. *Journal of Operations Management*, 27(2): 119-140 (2009).
6. Li S.H., Ragu-Nathanb B, Ragu-Nathanb T.S., Raob S.S. The impact of supply chain management practices on competitive advantage and organizational performance. *Omega*, 34(2): 107-124 (2006).
7. Huang X.W., Kristal M.M., Schroeder R.G. Linking learning and effective process implementation to mass customization capability. *Journal of Operations Management*, 26 (6): 714-729 (2008).
8. Lin H.F. Interorganizational and organizational determinants of planning effectiveness for Internet-based interorganizational systems. *Information & Management*, 43 (4): 423-433 (2006).
9. Modi S.B., Mabert V. A. Supplier development: Improving supplier performance through knowledge transfer. *Journal of Operations Management*, 25 (1): 42-64 (2007).
10. Mondragon A.E.C., Lyons A.C. Investigating the implications of extending synchronized sequencing in automotive supply chains: The case of suppliers in the European automotive sector.[J]. *International Journal of Production Research*, 46(11):2867-2888(2008).
11. Cousins P.D. The alignment of appropriate firm and supply strategies for competitive advantage. *International Journal of Operations & Production Management*, 25(5): 403-428 (2005).
12. Zimmer K.. Supply chain coordination with uncertain just-in-time delivery. *International Journal of Production Economics*, 77(1): 1–15 (2002).
13. Panayides P.M., So M. Logistics service provider–client relationships. *Transportation Research Part E*, 41(3): 179-200 (2005).
14. Kaynak H., Hartley J.L. Using replication research for just-in-time purchasing construct development. *Journal of Operations Management*, 24(6): 868-892 (2006).
15. Moore G.C., Benbasat I.. Development of an instrument to measure the perceptions of adopting an information technology innovation [J]. *Information Systems Research*, 2(2): 192–222 (1991).
16. Landis J.R., Koch C.G.. The measurement of observer agreement for categorical data. *Biometrics*, 33 (5): 159–174 (1977).
17. Fornell C., Larcker D.F. Evaluating structural equation models with unobservable variables and measurement error. *Journal of Marketing Research*, 18(1): 39-50 (1981).

An insertion heuristic for coordinated zoning paratransit systems

Chung-Wei Shen • Luca Quadrifoglio

1 Introduction

Paratransit systems are a demand responsive (DRT) type of service offering disabled customers a better service than fixed route transit systems because they provide curb-to-curb/door-to-door service and flexible schedules. The scheduling/routing of paratransit systems is commonly known as the dial-a-ride problem (DARP). Each passenger is transported by a ridesharing vehicle from a specific origin to a specific destination, at a desired departure or arrival time. DARP is a subclass of vehicle routing problems with pickups and deliveries (VRPPD), which can transport goods or persons. The volume of passengers using paratransit services have tremendously increased from the ADA (American with Disability Act) in 1990, which requires transit agency to provide disabled customers with a level of service equal to the one offered to regular ones. Paratransit services, however, are extremely costly to operate despite the ridesharing characteristic. In 2008, paratransit ridership made up only 1.8% of public transit ridership but 13.3% of the total operating cost in the United States [1].

For paratransit systems, a zoning strategy is a more practical method of operating the service due to their ever-sprawling and -expanding service areas. Utilizing a zoning strategy, service providers independently operate within their designated zones and can only cross into other regions to drop off their inter-zonal customers. For example, Metropolitan Transit System in San Diego employs a zoning strategy and divides its service area into four regions. In contrast, a centralized strategy considers a whole single region served by one designated provider (in Houston, for instance). A zoning strategy better fits locations where there is more than one regional center; a centralized strategy better fits places where there is one compact center. Because of increasing urban sprawl, a zoning strategy becomes more popular even if there is only one regional center. Other reasons for a zoning strategy include operating efficiency, customer service, and fares.

Utilizing a zoning strategy, however, will likely reduce the productivity of the system; in fact, additional geographical constraints (zone boundaries) are added to the system and the scheduling solution intuitively cannot be improved, as the set of feasible routing solutions is reduced. This is because the total vehicles empty backhaul miles (defined as the miles driven by a vehicle with no customers onboard, excluding the first/last trip segments to/from the depot) increase compared to the centralized strategy. Quadrifoglio, Dessouky & Ordonez [2]

found the operating choices of a zoning strategy to have a significant impact on the performance of demand responsive transit services.

A coordinated zoning system makes use of “transfer points” and is the operating practice adopted by some agencies (see paratransit in Chicago and the Twin Cities and rural agencies around Dallas, for example) requiring customers to switch vehicles at particular locations to complete their inter-zonal trips. This practice is attracting more and more attention from transit providers because of its perceived potential to significantly reduce operating costs mainly by reducing empty backhaul miles and by increasing rideshare rates. However, it requires synchronization between operators and increases customers’ discomfort, which might be quite undesirable for elderly or disabled riders (but certainly more tolerable for healthy customers, thus increasing the potential benefit of adopting this operating practice for regular demand responsive service within service areas). Coordination in this paper refers to switching inter-zonal passengers through the arrangement of the vehicle routes and schedules at specific transfer locations.

There is a need to quantify the trade-offs between the pros and cons when adopting the coordination strategy as opposed to not adopting it. As reviewed in the literature, coordination or integration between demand responsive transit systems and fixed-route transit systems has been investigated; however, the study of the coordination among independent zoning paratransit systems, especially for efficient heuristics, is regarded with little attention.

2 Literature review

Coordination of paratransit services increases not only efficiency and productivity but mobility. From the evaluation of Burkhardt [3], around \$700 million per year to transportation providers in the United States could be generated after implementing successful coordination. Consolidation of inter-zonal transportation will likely reduce trip costs by higher ridesharing rate and lower empty return miles [4]. The pickup and delivery problem (PDP) with transfer opportunity among vehicles was investigated by Shang & Cuff [5]. They proposed a heuristic procedure to solve the PDP in a health maintenance organization. However, this paper considered neither excess passenger travel times nor vehicle capacity constraints. An integer programming formulation of cooperative PDP with time windows was analyzed by Lin [6]. It concluded that the cooperative strategy may achieve savings in both total cost and vehicles used under the assumptions of all delivery locations are identical, the transfer is only allowed at the last pickup location of the returning vehicle, and the vehicle capacity is unlimited. Cortes, Matamala, & Contardo [7] showed the efficiency of transfer design of PDP in small instance. They suggested further developments of the transfer application on the “strategic design” and planning of paratransit system. Unlike DARP, PDP does not have travel times constraints to reduce customer dissatisfaction.

3 Systems definition

Within a demand response service area, passengers request a number of trips defined by the location and time. A transit provider may sub-divide the service area into zones. A zone is a geographical boundary. Each request can be categorized as inter-zonal requests or intra-zonal requests determined by the pick-up and drop location. Requests with pick-up and drop-off locations in different zones, are inter-zonal requests and requests with pick-up and drop-off locations in the same zone, are intra-zonal requests.

At every pick-up or drop-off node not situated at transfer location, we identify only one operation, which is either loading or unloading of passengers. At transfer locations, vehicles may either load or unload passengers or both. To recognize these operations, we generate two corresponding nodes (load node and unload node) at each transfer location for each inter-zonal request. When a vehicle visits the transfer node, it will either load or unload passengers according to the node’s characteristic.

Manhattan distances are used to calculate the symmetrical travel distance between any pairs of point. This estimated travel distance was verified to be close to the actual travel distance by Quadrifoglio et al. [2]. The distance calculation implies that the network is arranged in a grid pattern. We also assume no traffic jams on the system, and the travel time between two points is only a matter of travel distance and vehicle speed. This assumption may not allow considering precise travel time between points, but it does not alter the results of our following performance comparison. The link distances/speeds are input of the model and can be easily updated with more accurate values when available.

We assume that each inter-zonal request can only switch vehicles at a particular transfer location once, and each intra-zonal request do not switch vehicles to complete its trip. Concerning the customers' discomfort, more than one transfer might be quite undesirable other than quite unreasonable. In fact, in practice, the passengers of paratransit are assured to transfer at most once in Chicago and Boston (both using a coordinated zoning system).

4 Algorithm description

The insertion algorithm is used to schedule the proposed services. This algorithm processes ride requests sequentially, inserting one customer at a time into the vehicle schedule until all requests have been processed. At the beginning of insertion algorithm, one empty route leaves and ends at each depot is generated. The intra-zonal requests will be scheduled by classic insertion procedure on DARP. Every inter-zonal request generates a drop-off and a pick-up node at a transfer location. The request can be treated as two intra-zonal requests where schedules are coordinated such that the inter-zonal customers can switch vehicles at transfer locations. According to the designated zone of each request, the schedule is checked for possible insertion of every unassigned request sequentially by their earliest pick-up time. All unassigned requests need to search the feasible insertion slots from these routes that minimize the vehicle travel distance. During the procedure of searching feasible slot, four constraints are taking into consideration. First, for each customer the drop-off time should be always later than its pick-up time. Second, the unassigned trips can only be assigned into the time slots within their pick-up and drop-off time windows. Third, after inserting the new trip, we still need to check whether if this insertion will violate the successive assigned customers' time windows. Finally, the capacity of vehicle is needed to consider in the process of inserting the unassigned trips. Once a feasible insertion is selected, the variables for each assigned location need to be updated. We maintain two quantities for each assigned node: waiting time and the maximal postponed time interval. These two quantities increase the efficiency of inserting the new requests. In our algorithm, we allow both non-empty and empty load vehicles to wait at pick-up location before the ready service time. This assumption can increase the possibility of feasible insertion when operating the algorithm. Fig. 1 illustrates the algorithm procedure in diagram. Comparison of the results computed with the heuristic to those obtained from the CPLEX in small instances which can be optimized in reasonable time shows that the heuristic procedure is able to produce high-quality solutions. This paper investigates the performance of coordinated zoning strategy as opposed to alternative strategies by applying the Metropolitan Transit Authority of Harris County demand response service data set. At the conference we will further present any control parameters and solution procedure we have considered.

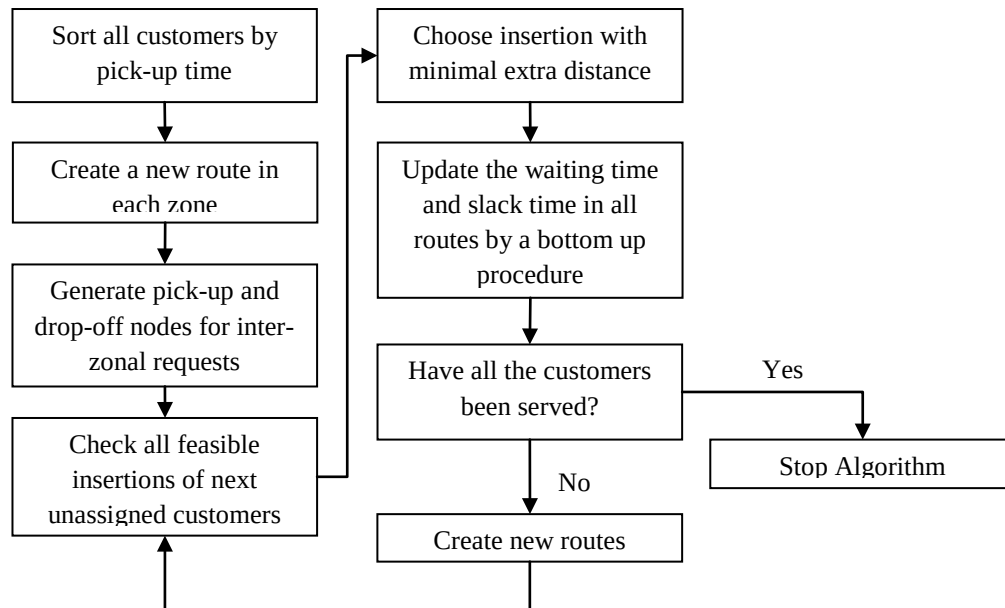


Fig. 1 Flow chart of the algorithm

References

1. 2010 public transportation fact book, (61st ed.), Washington, DC: American Public Transportation Association (2010).
2. Quadrifoglio, L., Dessouky, M. M., & Ordonez, F., A simulation study of demand responsive transit system design. *Transportation Research Part A: Policy and Practice*, 42(4), 718-737 (2008).
3. Burkhardt, J. E., Economic benefits of coordinating human service transportation and transit services. *Transportation Research Record: Journal of the Transportation Research Board*, 1887, 55-61 (2004).
4. Cook, T. J., Lawrie, J. J., & Henry, A. J., From rural single-county to multicounty regional transit systems: Benefits of consolidation. *Transportation Research Record: Journal of the Transportation Research Board*, (1841), 54-61 (2003).
5. Shang, J. S., & Cuff, C. K., Multicriteria pickup and delivery problem with transfer opportunity. *Computers and Industrial Engineering*, 30(4), 631-645 (1996).
6. Lin, C. K. Y., A cooperative strategy for a vehicle routing problem with pickup and delivery time windows. *Computers and Industrial Engineering*, 55(4), 766-782 (2008).
7. Cortes, C. E., Matamala, M., & Contardo, C., The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research*, 200(3), 711-724 (2010).

Improving Schedule Stability in Single-Machine Rescheduling due to New Operation Insertion

Can Akkan

1 Introduction

In this research we address a scheduling problem in which a new operation is to be inserted into an existing schedule for a single machine. We assume this is a finite-capacity schedule that has been prepared for capacity reservation and due-time commitment. Insertion of a new operation can easily trigger extensive changes in the existing schedule. Therefore maintaining a stable schedule, while making room for a new operation is an important concern. In the current model, the existing schedule, or the pre-schedule, is assumed to have $(n - 1)$ operations such that each operation i has a release-time, r_i , start-time, s_i^o , processing time, p_i and a due-time, d_i . It is assumed that for each i , $r_i \leq s_i^o$ and $s_i^o + p_i \leq d_i$. Furthermore, the existing schedule has inserted idle-times, or “gaps” between scheduled operations, in order to have some robustness in the schedule. The new operation being inserted into the schedule is indexed with n , and just like other operations it has a release-time, r_n , due-time, d_n and processing time, p_n .

It is further assumed that the insertion is going to be taking place in the “near-term” portion of the schedule, so that although the existing schedule is not frozen, any rescheduling that is done should not create a ripple effect towards “now”. Therefore, rescheduling is not allowed to violate the release-time constraint of an operation.

Letting S_i denote the start-time of operation $i = 1, \dots, n$ after the insertion of new operation, the objective of the rescheduling is minimization of maximum tardiness of operations $i = 1, \dots, n$, (let tardiness be denoted by $T_i = \max(0, S_i + p_i - d_i)$). On the other hand, a secondary objective that measures schedule stability is average absolute deviation of start-times, denoted and calculated as, $\overline{AD} = \sum_{i=1}^{(n-1)} |s_i^o - S_i| / (n - 1)$.

2 The Solution Approach

The solution approach proposed can best be described as a branch-and-bound directed heuristic search that is designed to find a schedule with minimum maximum tardiness,

Can Akkan
Sabancı University, Sabancı School of Management, Istanbul, Turkey
E-mail: canakkan@sabanciuniv.edu

say T_{max}^* and among alternative schedules with $T_{max} = T_{max}^*$ one that has the lowest $\overline{AD}$. There are two reasons why this algorithm is a heuristic: Firstly, modified, or dummy, due-times are used as input in order to direct the algorithm to find solutions that yield more stable schedules. These dummy due-times, denoted by d'_i are determined by different heuristic algorithms such that $d'_i \in [s_i^o + p_i, d_i]$. Intuitively speaking, by setting some operations to have tighter d'_i given as input to the branch-and-bound algorithm, solutions where those operations are not scheduled very differently from their initial schedule are favored. Therefore, letting $T'_i = \max(0, S_i + p_i - d'_i)$, the algorithm finds a schedule that minimizes T'_{max} , but not necessarily T_{max} . Secondly, the dominance rules used in the branch-and-bound algorithm are for minimization of maximum tardiness, which may lead to sequences that minimize $\overline{AD}$ to be removed from the search. Therefore, for each full-sequence generated by the branch-and-bound an adjacent pairwise interchange heuristic is run to find a schedule that improves $\overline{AD}$ without increasing T'_{max} .

2.1 The Branch-and-Bound Algorithm

The branch-and-bound algorithm makes use of Allen's, [1], interval algebra, Esquirol et al.'s, [4], interval structures, a dominance theorem by Erschler et al., [3], and bounds on start-times of operations by Briand et al., [2]. These are used to go through a solution space that only includes non-dominated sequences for the minimization of T'_{max} . It is important to note that for a given partial sequence in the branch-and-bound tree a lower bound on T'_{max} (T'_{max-LB}) and a lower bound on $\overline{AD}$ ($\overline{AD}_{LB}$) are calculated. At any given node (including the root node) there is an upper bound on T'_{max} (T'_{max-UB}) and an upper bound on $\overline{AD}$ ($\overline{AD}_{UB}$). A node is fathomed if (i) $T'_{max-UB} < T'_{max-LB}$ or (ii) $T'_{max-UB} = T'_{max-LB}$ and $\overline{AD}_{UB} < \overline{AD}_{LB}$.

2.1.1 Basic concepts

Definition 1: An interval structure is a couple $\langle I, C \rangle$ with $I = \{i_1, \dots, i_n\}$ a set of intervals and C a set of constraints over $I \times I$. Each interval i_j is defined by its lower and upper bounds x_j and y_j , respectively.

Any constraint between two intervals can be expressed by using the relations of the interval algebra proposed by [1]. We will be using the relation $during(A, B)$, which is depicted in Figure 1.

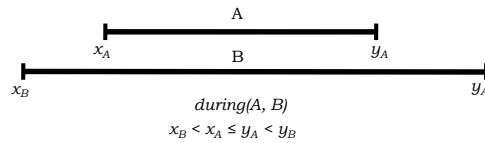


Fig. 1 Allen's relation *during*

Definition 2: A top of an interval structure $\langle I, C \rangle$ is an interval $t \in I$ such that $\forall i \in I$ the Allen's relation *during*(i, t) never holds.

In Figure 2, tops are intervals $t_1 = 1$ and $t_2 = 3$.

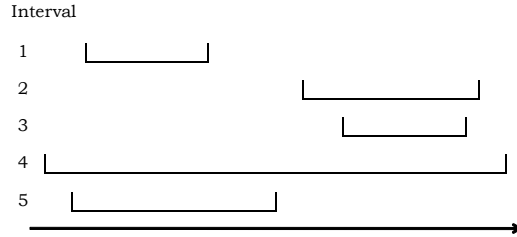


Fig. 2 Tops

The notion of a top is used to define a t-pyramid as follows:

Definition 3: Given a top t_α , a t-pyramid P_α related to t_α is the set of intervals $i \in I$ such that *during*(t_α, i) holds.

Figure 3 gives the pyramid related to $t_1 = 1$: $P_1 = \{4, 5\}$.

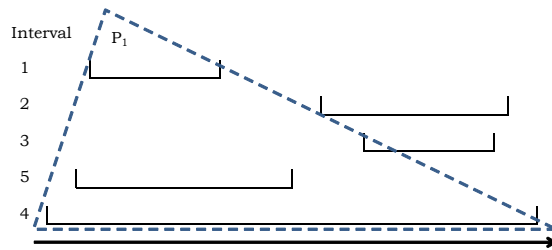


Fig. 3 Pyramid of a top

Given these definitions, [2] has developed the following dominance theorem for $1/r_j/Lmax$ and $1/r_j/Tmax$ that defines a partial order. The interval structure $\langle I, C \rangle$ associated with a given problem contains an interval $i_j = [r_j, d_j] \in I$ for each job j . It is assumed that the tops are indexed in ascending order of their release-times, r_j , or in case of ties, in ascending order of their due-times. When there is tie in both release-times and due-times, then tops are indexed in arbitrary order. The function $u(j)$ (resp., $v(j)$) indicates the index of the first (resp., last) t-pyramid to which the job interval i_j belongs.

Theorem 1: A dominant set of sequences can be constituted by the sequences such that:

1. the tops are in ascending order with respect to their index;
2. only the jobs belonging to the first pyramid can be located before the first top and the are in ascending order of their release-times (or in arbitrary order in case of ties);

3. only the jobs belonging to the last pyramid can be located after the last top and they are in ascending order of their due-times (or in arbitrary order in case of ties);
4. only the jobs belonging to the t-pyramids P_k and P_{k+1} can be located between two successive tops t_k and t_{k+1} so that:
 - first the jobs belonging only to P_k but not to P_{k+1} are sequenced immediately after t_k in ascending order of their due-times (or in arbitrary order in case of ties),
 - then the jobs belonging to both P_k and P_{k+1} are sequenced in an arbitrary order,
 - and last the jobs belonging only to P_{k+1} but not to P_k are sequenced in ascending order of release-times (or in arbitrary order in case of ties).

Let S_{dom} be the set of non-dominated sequences for a given instance.

Figure 4 gives the instance that will be used to demonstrate the algorithms discussed in this paper. Note that Job 5 is the new job being inserted into the schedule.

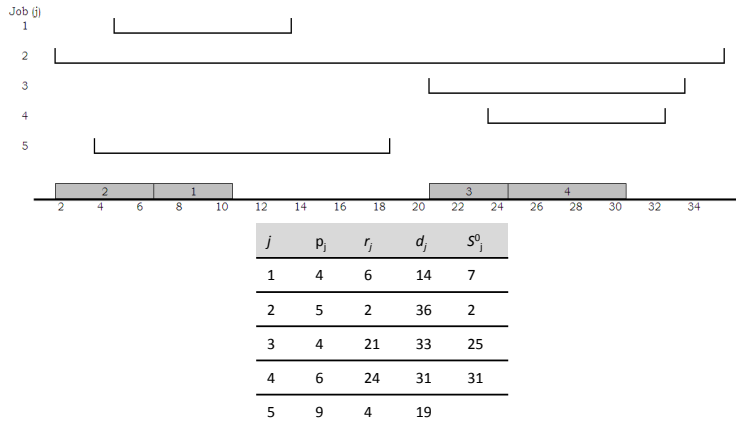


Fig. 4 Example

2.1.2 Branching scheme

The root of the branch-and-bound tree has the partial sequence for the operations corresponding to the tops of the interval structure. Then at each level l , such that $l \geq 1$, added below the root, operations that can be scheduled before top t_l are added to the partial sequence so that the resultant partial sequence is not dominated (based on Theorem 1). If there are m tops, at level m of the tree, in addition to operations sequenced before t_m , operations after t_m are also added to the sequence so that full sequences are achieved at this level.

Figure 5 partially shows the three levels of the branch-and-bound tree for the example, without using any bounds, thus essentially doing an enumeration of all non-dominated sequences. Note that, for $n = 5$, there are a total $5! = 120$ sequences, out of which only 12 are enumerated with this tree.

level	description	branch	sequence
0	sequence the tops	0	$1 \prec 4$
1	add tasks that may precede t_1	0.1	$1 \prec 4$
		0.2	$5 \prec 1 \prec 4$
		0.3	$2 \prec 1 \prec 4$
		0.4	$2 \prec 5 \prec 1 \prec 4$
2	add tasks between t_1 and t_2	0.1.1	$1 \prec 5 \prec 4$
		0.1.2	$1 \prec 5 \prec 2 \prec 4$
		0.1.3	$1 \prec 5 \prec 3 \prec 4$
		0.1.4	$1 \prec 5 \prec 2 \prec 3 \prec 4$
		0.2.1	$5 \prec 1 \prec 4$
		0.2.2	$5 \prec 1 \prec 2 \prec 4$
		0.2.3	$5 \prec 1 \prec 3 \prec 4$
		0.2.4	$5 \prec 1 \prec 2 \prec 3 \prec 4$
		$\vdots$	
		$\vdots$	
3	add tasks after t_2	0.1.1.1	$1 \prec 5 \prec 4 \prec 3 \prec 2$
		0.1.2.1	$1 \prec 5 \prec 2 \prec 4 \prec 3$
		$\vdots$	

Fig. 5 Structure of the branch-and-bound tree

2.1.3 Bounds on job completion times

Briand et al., [2], define two sequences that lay the foundation for calculating bounds. The most favorable sequence $Seq_j^{min} \in S_{dom}$ for j is the sequence in which j is completed as early as possible. Similarly, the most unfavorable sequence $Seq_j^{max} \in S_{dom}$ for j is the sequence in which j is completed as late as possible. The following two theorems of [2] describe how Seq_j^{min} and Seq_j^{max} can be determined:

Theorem 2: (Theorem 2 of [2]) The most favorable sequence $Seq_j^{min} \in S_{dom}$ for j is $\sigma_1 \prec t_1 \prec \dots \prec \sigma_{u(j)-1} \prec t_{u(j)-1} \prec j$ so that the jobs of σ_k are sequenced in ascending order of their release-times, where $\sigma_k = \{i \in Pred_j^{min} | u(i) = k\}$ and $Pred_j^{min}$ is the set of jobs such that $v(k) < u(j)$.

Theorem 3: (Theorem 4 of [2]) The most unfavorable $Seq_j^{max} \in S_{dom}$ for j is the sequence $t_1 \prec \sigma_1 \prec \dots \prec t_{v(j)-1} \prec \sigma_{v(j)-1} \prec A \prec t_{v(j)} \prec B \prec j$ so that jobs of σ_k are sequenced in ascending order of their due-times, the jobs of A are sequenced in ascending order of their release-times, and the jobs of B are sequenced in ascending order of their due-times, where σ_k , A and B are defined as follows:

$$\begin{aligned} \sigma_k &= \{i \in Pred_j^{max} | v(i) < v(j) \text{ and } v(i) = k\}, \\ A &= \{i \in Pred_j^{max} | v(i) \geq v(j) \text{ and } d_i > d_j\}, \\ B &= \{i \in Pred_j^{max} | v(i) \geq v(j) \text{ and } d_i \leq d_j\}, \text{ such that} \\ &Pred_j^{max} = \{k | u(k) \leq v(j)\}. \end{aligned}$$

In the example, there are two tops, $t_1 = 1$ and $t_2 = 4$ with associated pyramids, $P_1 = \{2, 5\}$, $P_2 = \{2, 3\}$. For the example, Seq_j^{min} and Seq_j^{max} are determined as follows:

Given Seq_j^{min} (resp. Seq_j^{max}) one can easily calculate T_j^{min} (resp. T_j^{max}), the best (resp. worst) tardiness of a job j among all the sequences of S_{dom} . However, to be

j	$u(j)$	$Pred_j^{min}$	Seq_j^{min}	$v(j)$	$Pred_j^{max}$	Seq_j^{max}
1	1	$\emptyset$	1	1	$\{2, 5\}$	$2 \prec 5 \prec 1$
2	1	$\emptyset$	2	2	$\{1, 3, 4, 5\}$	$1 \prec 5 \prec 4 \prec 3 \prec 2$
3	2	$\{1, 5\}$	$5 \prec 1 \prec 3$	2	$\{1, 2, 4, 5\}$	$1 \prec 5 \prec 2 \prec 4 \prec 3$
4	2	$\{1, 5\}$	$5 \prec 1 \prec 4$	2	$\{1, 2, 3, 5\}$	$1 \prec 5 \prec 2 \prec 3 \prec 4$
5	1	$\emptyset$	5	1	$\{1, 2\}$	$2 \prec 1 \prec 5$

Table 1 Seq_j^{min} and Seq_j^{max} for the example

able to determine T_j^{min} and T_j^{max} at a given node of the branch-and-bound tree, it is necessary to introduce a few more definitions.

Definition 4: Let Π_n be the partial sequence at node n of the branch-and-bound tree, and $l(n)$ be the level of the node (assume root of the tree is level 0). Then, $\Pi_n = P_1 \prec t_1 \prec P_2 \prec t_2 \prec \dots \prec P_{l(n)} \prec t_{l(n)} \prec t_{l(n)+1} \prec \dots \prec t_m$, where m is the number of tops in the corresponding interval structure.

Definition 5: Let $\Pi_{n,[l]}$ be the partial sequence at node n of the branch-and-bound tree up to and including t_l , i.e., $\Pi_{n,[l]} = P_1 \prec t_1 \prec P_2 \prec t_2 \prec \dots \prec P_l \prec t_l$.

Corollary 1: $Pseq_j^{min}$, the most favorable partial sequence for Job j at a branch-and-bound node n with partial sequence Π_n , is determined as follows,

- Case 1: $l(n) \geq u(j) - 1$
If $j \in \Pi_{n,[l(n)]}$ then let $Pseq_j^{min} = \Pi_n$ up to and including j , otherwise let $Pseq_j^{min} = \Pi_{n,[l(n)]} \prec Pred_j^{min} \setminus \Pi_{n,[l(n)]} \prec j$, where jobs in $Pred_j^{min} \setminus \Pi_{n,[l(n)]}$ are sequenced in increasing due-times.
- Case 2: $l(n) < u(j) - 1$
Let $I = (\sigma_1 \cup \sigma_2 \cup \dots \cup \sigma_{(u(j)-1)}) \cap (P_1 \cup P_2 \dots \cup P_l)$, then, $Pseq_j^{min} = \Pi_{n,[l(n)]} \prec (\sigma_l \setminus P_l) \cup (\sigma_{l+1} \setminus I) \prec t_{l+1} \prec \dots \prec (\sigma_{u(j)-1} \setminus I) \prec t_{u(j)-1} \prec j$, where σ_i are determined according to Theorem 1, and jobs in $(\sigma_l \setminus P_l) \cup (\sigma_{l+1} \setminus I)$ and in $(\sigma_{u(j)-1} \setminus I)$ are sequenced in ascending order of release-times.

In the example, $Pseq_j^{min}$ for node 0.2 with $\Pi_{0.2} = 5 \prec 1 \prec 4$ are calculated as shown in Table 2, resulting in $T_{max}^{lb} = 3$:

j	$Pseq_j^{min}$	S_j^{min}	C_j^{min}	T_j^{min}
1	$5 \prec 1$	13	17	3
2	$5 \prec 1 \prec 2$	17	22	0
3	$5 \prec 1 \prec 3$	21	25	0
4	$5 \prec 1 \prec 4$	24	30	0
5	5	4	13	0

Table 2 $Pseq_j^{min}$ and C_j^{min} at node 0.2 in the example

Corollary 2: $Pseq_j^{max}$, the most unfavorable partial sequence for Job j at a branch-and-bound node n with partial sequence Π_n , is determined as follows,

- Case 1: $l(n) \geq v(j)$
If $j \in \Pi_{n,[l(n)]}$ then let $Pseq_j^{max} = \Pi_n$ up to and including j , otherwise let $Pseq_j^{max} = \Pi_{n,[l(n)]} \prec Pred_j^{max} \setminus \Pi_{n,[l(n)]} \prec j$, where jobs in $Pred_j^{max} \setminus \Pi_{n,[l(n)]}$ are sequenced in ascending order of their due-times.

- Case 2: $1 \leq l(n) < v(j)$

Let $I = (\sigma_1 \cup \sigma_2 \cup \dots \cup \sigma_{v(j)-1}) \cap (P_1 \cup P_2 \dots \cup P_l)$. If $j \in \Pi_n$ then let $Pseq_j^{max} = \Pi_n$ up to and including j , otherwise let $Pseq_j^{max} = \Pi_{n, [l(n)]} \prec \sigma_l \setminus I \prec t_{l+1} \prec \dots \prec \sigma_{v(j)-1} \setminus I \prec A \setminus I \prec t_{v(j)} \prec B \setminus I \prec j$, where σ_i are determined according to Theorem 1. $A \setminus I$ are sequenced in increasing order of their release-times and $B \setminus I$ are sequenced in increasing due-times.

- Case 3: $l(n) = 0$

$$Pseq_j^{max} = Seq_j^{max}$$

In the example, $Pseq_j^{max}$ for node 0.2 with $\Pi_{0.2} = 5 \prec 1 \prec 4$ are calculated as shown in Table 3, resulting in $T_{max}^{ub} = 3$.

j	$Pseq_j^{max}$	S_j^{max}	C_j^{max}	T_j^{max}
1	$5 \prec 1$	13	17	3
2	$5 \prec 1 \prec 4 \prec 3 \prec 2$	34	39	3
3	$5 \prec 1 \prec 2 \prec 4 \prec 3$	30	34	1
4	$5 \prec 1 \prec 2 \prec 3 \prec 4$	24	30	0
5	5	4	13	0

Table 3 $Pseq_j^{max}$ and C_j^{max} at node 0.2 in the example

Given the S_j^{min} and S_j^{max} calculated for a given node, it is quite straightforward to calculate a lower bound on $\overline{AD}$, $\overline{AD}_{LB} = 1/(n-1) \sum_1^{n-1} AD_j^{lb}$, where

$$AD_j^{lb} = \begin{cases} 0, & \text{if } S_j^{min} \leq S_j^0 \leq S_j^{max} \\ S_j^{min} - S_j^0, & \text{if } S_j^{min} > S_j^0 \\ S_j^0 - S_j^{max}, & \text{if } S_j^{max} < S_j^0 \end{cases}$$

Table 4 shows calculation of AD_j^{lb} at node 0.2 in the example, resulting in $\overline{AD}_{LB} = 5.5$.

j	S_j^0	S_j^{min}	S_j^{max}	D_j^{lb}
1	7	13	13	$13 - 7 = 6$
2	2	17	34	$17 - 2 = 15$
3	21	21	30	0
4	25	24	24	$25 - 24 = 1$

Table 4 AD_j^{lb} calculations at node 0.2 in the example

2.1.4 Minimizing $\overline{AD}$ Given A Sequence

This algorithm, referred to as **MINAD**, minimizes the $\overline{AD}$ given a sequence. Therefore, it is used within heuristics to create a schedule from a given sequence.

At first, given a sequence, say σ , an earliest-start schedule is created. Then, a backward scheduling algorithm delays operations to improve $\overline{AD}$ while not increasing T_{max}' . When a pass is completed without delaying any operation to improve $\overline{AD}$, the heuristic stops; otherwise another backward pass is done. Delaying decision is made

as follows. Let $\sigma_{[k]}, \sigma_{[k+1]}, \dots, \sigma_{[l]}$ for $k \leq l$, be a partial sequence of consecutive operations such that $C_{[l]} < S_{[l+1]}$, that is, there is a gap after $\sigma_{[l]}$, whereas $C_{[j]} = S_{[j+1]}$ for all $j = k, \dots, l-1$. Letting $D_{[j]} = S_{[j]} - s_{[j]}^0$, k is the largest index less than or equal to l such that $D_{[k]} < 0$ and,

$$|\{j : D_{[j]} < 0, k \leq j \leq l, \sigma_{[j]} \neq n\}| = |\{j : D_{[j]} \geq 0, k \leq j \leq l, \sigma_{[j]} \neq n\}| + 1$$

Operation $\sigma_{[k]}$ for which the above criteria are met can be delayed to reduce $\overline{AD}$. The amount of delay is determined as the minimum of the following quantities:

$$\begin{aligned} (i) & S_{[l+1]} - C_{[l]} \\ (ii) & \max \{D_{[j]} : D_{[j]} < 0, k \leq j \leq l, \sigma_{[j]} \neq n\} \\ (iii) & \min_j \{T'_{max} + d'_{[j]} - C_{[j]} : k \leq j \leq l\} \end{aligned}$$

where (ii) ensures that no operation, j , that is currently starting before s_j^0 is delayed beyond s_j^0 , and (iii) ensures that no operation is delayed to increase T'_{max} .

2.1.5 An Upper Bound Heuristic

The following construction heuristic is used to determine a good upper bound at the root of the branch-and-bound tree. Recall that $T_i = \max(0, S_i + p_i - d_i)$, whereas $T'_i = \max(0, S_i + p_i - d'_i)$. Since the heuristic is making use of biased random sampling, it is used within an iteration loop and the best solution found in these iterations is reported. Letting m denote the number of tops for the intervals defined by r_i and d'_i for $i = 1, \dots, n$, the number of iterations is set to be $(n+1)(n-m)+1$. Note that, if $n = m$ only one iteration is done and it gives the sequence of tops in increasing top index.

1. Sequence the tops in increasing top index, create an earliest-start schedule, and then right-shift them to obtain the initial partial schedule without increasing T'_j for any j .
2. Sort non-top jobs in increasing $(d'_j - r_j - p_j)$, if there is a tie choose arbitrarily. Let index $L[i]$ denote the index of the job at the i^{th} position in the sorted list.
3. Let σ denote the partial schedule at a given iteration of the algorithm, with $\sigma_{[k]}$ denoting the k^{th} job in σ . Furthermore, let $S_{[k]}$ and $C_{[k]}$ be the start-time and completion-time of $\sigma_{[k]}$.
4. For $i = 1$ to $n - m$
 - (a) Let g_k be the gap between $\sigma_{[k]}$ and $\sigma_{[k+1]}$, defined by the limits $(C_{[k]}, S_{[k+1]})$, for $k = 0, \dots, (m+1+i)$, where $C_{[0]} = 0$ and $S_{[n+1]} = \infty$.
 - (b) Let λ_k denote the maximum left-slide that can be done on $\sigma_{[k]}$, for $k = 1, \dots, (m+1+i)$. Then, assuming $\lambda_0 = 0$, $\lambda_k = \min(\lambda_{k-1} + S_{[k]} - C_{[k-1]}, S_{[k]} - r_{[k]})$.
 - (c) Let $top(k)$ be the first top to the left of gap k .
 - (d) Let $rl(j, k)$ be the right-limit of the k^{th} gap for operation j , i.e. $\min(d'_j, S_{[k+1]})$.
 - (e) Let $ll(j, k)$ be the left-limit of the k^{th} gap for operation j , i.e. $\max(r_j, C_{[k]})$.
 - (f) Let, γ_k be the effective size of non-dominated gaps for $k = 0, \dots, (m+1+i)$,

$$\gamma_k = \begin{cases} \max(0, rl(L[i], k) - ll(L[i], k)) & \text{if } u(L(i)) - 1 \leq top(k) \leq v(L(i)) \\ -\infty & \text{otherwise} \end{cases}$$

- (g) Let τ_k be the maximum tardiness w.r.t. d'_j for the current partial schedule if $L(i)$ is inserted into gap k by “pushing” the successors if necessary. For gaps k with $\gamma_k = -\infty$, let $\tau_k = \infty$ for others τ_k is calculated as follows:
 - Calculate T'_j for operations scheduled before gap k , let T'_{pred} be the maximum tardiness of these operations.
 - Determine the earliest feasible start-time for operation $L(i)$ in gap k , as $S_k^{min} = \max \{C_{[k]} - \lambda_k, r_{[k]}\}$ and determine the right-shift required to push its immediate successor to maintain feasibility.
 - If right-shift in the previous step is positive continue right-shifting successors until a feasible schedule achieved, while doing this calculate the T'_j for these operations.
 - (h) Let $\delta_k = |s_{L(i)}^o - S_k^{min}|$
 - (i) Let $\tau_{min} = \{\tau_k : k = 0, \dots, (m+1+i)\}$. The gap in which operation $L(i)$ will be inserted is selected from among the gaps for which $\tau_k = \tau_{min}$ using biased random sampling. The probability distribution to do this is generated by using δ_k , so that gaps with small δ_k are more likely to be selected. Let g be the gap that is selected randomly.
 - (j) If $\min(d_{L[i]}, S_{[m+1]}) - \max(r_{L[i]}, C_{[m]}) \geq p_{L[i]}$ then insert job $L[i]$ with $S_{L[i]} = S_{[m+1]} - p_{L[i]}$, that is, as late as possible inside gap g .
 - (k) Otherwise, slide operations before gap g to the left in order to increase the gap size to as close to $p_{L[i]}$ as possible. If gap size is still less than $p_{L[i]}$ then slide operations after gap g (hence increase tardiness) until gap size is equal to $p_{L[i]}$. Then insert operation $L[i]$ into gap g .
5. Use MINAD to minimize $\overline{AD}$ of the schedule.
 6. Apply adjacent pairwise interchange heuristic (API), which is described below.

2.1.6 Adjacent Pairwise Interchange Heuristic

Given a schedule, σ , this heuristic, API, attempts to improve $\overline{AD}$ by iteratively interchanging two adjacent operations without increasing T'_{max} of the given schedule. The heuristic starts from the left-most operation in the sequence, moving forward. When interchanging operations $\sigma_{[k]}$ and $\sigma_{[k+1]}$, the start-times of operations $\sigma_{[k-1]}$ and $\sigma_{[k+1]}$ are not changed. When a pass is completed, that is an attempt is made to interchange all adjacent pairs of operations and no improvement in $\overline{AD}$ has been achieved, then the iterations are ended, otherwise another forward-pass iteration is done. As the final step of the heuristic MINAD is run.

2.2 Due-time Adjustment Heuristics

There are two benchmark heuristics and eleven heuristics that determine, d'_i , by ranking each operation in increasing order of a certain criterion. Let $\rho_i, i = 1, \dots, n-1$, be that rank. Then, $d'_i = (\kappa_i/n)[d_i - (s_i^o + p_i)] + (s_i^o + p_i)$, where, $\kappa_i \in \{0, \rho_i, n\}$. Therefore, heuristics differ by the criterion used to determine ρ_i . The two benchmark heuristics are as follows:

- H1: $\kappa_i = n$, i.e. $d'_i = d_i$.
- H2: $\kappa_i \sim U(0, n)$.

The first tested heuristic simply sets the dummy due-time to be the pre-schedule completion time.

H3: $\kappa_i = 0$, i.e. $d'_i = s_i^o + p_i$.

A group of heuristics, which are referred to as fragmentation-based heuristics, analyze the gaps in the pre-schedule within $[r_i, d_i]$ window for each operation $i = 1, \dots, (n-1)$, to determine average gap size, number of gaps, and range of gap sizes. Then, rank ρ_i of operation is equal to k such that operation i has the k^{th} smallest,

H4: range of gap sizes.

H5: number of gaps.

H6: average gap size.

The common feature of the other group of heuristics is that any operation, i , whose time window $[r_i, d_i]$ does not overlap with that of the new operation, is assigned $\kappa_i = 0$. For other operations, rank ρ_i of operation is equal to:

H7: k such that $(d_i - r_i)/p_i$ is the k^{th} smallest of $(d_l - r_l)/p_l$ for $l = 1, \dots, n-1$.

H8: k such that $-|p_i - p_n|$ is the k^{th} smallest of $-|p_l - p_n|$ for $l = 1, \dots, n-1$.

H9: n for all i

The final group of heuristics uses the overlap, o_i , between $[r_i, d_i]$ and $[r_n, d_n]$, i.e. $o_i = \min(d_n, d_i) - \max(r_n, r_i)$. There are four heuristics within this group. The first two sets $\rho_n = n$ and for $i = 1, \dots, n-1$, sets rank ρ_i of operation i equal to k such that:

H10: o_i is the k^{th} smallest of $\{o_l : o_l > 0, l = 1, \dots, n-1\}$, and $\rho_i = 0$ if $o_i = 0$.

H11: o_i/p_i is the k^{th} smallest of $\{o_l/p_l : o_l > 0, l = 1, \dots, n-1\}$, and $\rho_i = 0$ if $o_i = 0$.

The other two, includes the new operation in the ranking process directly, such that, rank ρ_i of operation i is equal to k such that:

H12: o_i is the k^{th} smallest of $\{o_l : o_l > 0, l = 1, \dots, n\}$, and $\rho_i = 0$ if $o_i = 0$.

H13: o_i/p_i is the k^{th} smallest of $\{o_l/p_l : o_l > 0, l = 1, \dots, n\}$, and $\rho_i = 0$ if $o_i = 0$.

3 Preliminary Results

3.1 Instance Generation

Firstly, a pre-schedule is generated using two parameters: ϱ , that affects the spread of release-times, and τ , that affects the tightness of due-time. p_i is randomly generated from $DU(1, 100)$. Letting $P_{tot} = \sum_{i=1}^{n-1} P_i$, $r_i \sim DU(0, \varrho P_{tot})$, where $\varrho \in \{0.50, 0.75, 1.00\}$, and $d_i \sim DU(d_{min}^i, \tau P_{tot})$, where $\tau \in \{1.25, 1.50, 1.75\}$, and d_{min}^i is determined so that there is at least one feasible gap for operation i to be scheduled in. Once r_i and d_i are known, operations are inserted in increasing order of i so that first a feasible gap is randomly selected and then a feasible start-time, s_i^o , is selected randomly within that gap. Secondly, a new operation is generated using two parameters: λ affects the length of the new operation relative to the available gap sizes in the pre-schedule, and δ that affects the tightness of its due-time. After finding all the gaps in the pre-schedule, one of the gaps is selected randomly. If the size of that gap is g , then p_n is set to be $\max(g + 1, \lambda g)$, where $\lambda \in \{1.10, 1.15, 1.20\}$. Then, r_n and d_n are

determined so that there is no gap within the interval $[r_n, d_n]$ that is greater than or equal to p_n . For each combination of $\varrho, \lambda, \tau, \delta$ 10 random instances are generated.

Since the instance generation procedure is rather involved we present some summary statistics regarding these instances in Table 5 and Table 6, where $w_i (= d_i - r_i)$ is the size of the window within which operation i should be inserted.

Table 5 Summary Statistics of Instances - Part 1

				gap size					
	Cmax	util	#gaps	min	max	ave	p_n	w_n	p_n/w_n
n=20									
min	761	0.56	3	1	8	4	18	126	2.8%
max	1788	0.97	13	30	403	106.9	100	1432	66.7%
ave	1241.2	0.72	8.86	3.9	145.3	40.6	83.8	718.8	17.9%
median	1221.0	0.72	9.00	2.0	131.0	38.1	87.5	814.5	10.2%
std dev	208.3	0.08	1.92	4.9	76.1	16.5	15.3	345.1	15.3%
CV	0.17	0.11	0.22	1.25	0.52	0.41	0.18	0.48	0.85

These statistics show that there is reasonable variation in the relevant characteristics (namely, C_{max} , utilization, number of gaps, pocessing time and the window of new operation).

Table 6 Summary Statistics of Instances - Part 2

	$w_i, i = 1, \dots, (n-1)$			$p_i/w_i, i = 1, \dots, (n-1)$		
	min	max	ave	min	max	ave
n=20						
min	10.00	602.00	361.30	0.1%	11.6%	6.4%
max	393	1951	1001.65	5.3%	94.9%	19.8%
ave	141.52	1216.20	641.30	1.0%	41.5%	11.1%
median	135.50	1183.50	617.83	0.8%	38.1%	10.7%
std dev	69.05	252.69	139.75	0.8%	18.0%	2.6%
CV	0.49	0.21	0.22	81.1%	43.5%	23.7%

3.2 Results

The results are obtained for $n = 20$ and reported relative to Heuristic 1 ($d'_i = d_i$) and Heuristic 2 (random). Let, $\overline{AD}_{h,k}$ and $MT_{h,k}$ denote the average absolute deviation in start-times, and maximum tardiness with respect to due-time d_i , respectively, for heuristic h and instance k . Then, we define $\Delta\overline{AD}_{i,h,k} = \overline{AD}_{i,k} - \overline{AD}_{h,k}$, and $\Delta MT_{i,h,k} = MT_{i,k} - MT_{h,k}$ for $i = 1, 2$. Table 7 shows the averages of these paired differences. Since these paired differences were not normally distributed, to test for statistical significance, Wilcoxon signed rank test was used (the non-paramteric equivalent of the paired t-test) with the statistical package R. Each entry in Table 7 is an average of 450 test instances. There were a total of 5850 instances solved which required an average CPU time of 3.9363 seconds. The algorithms were coded in C programming

language and run on a personal computer with Interl Core i5 2.53 GHz CPU and 3 GB RAM. A stopping condition of a maximum 250,000 nodes was imposed on the implementation of the branch-and-bound algorithm.

In testing the performance of heuristics with respect to T_{max} relative to H1 (the first column in Table 7) the null hypothesis was “difference is less than or equal to 0”, in other words the heuristic does not lead to a larger T_{max} relative to H1. Null hypothesis was rejected for all but H9, H10, and H12, leading us to the conclusion that these three heuristics do not result in any significant increase in T_{max} . To conclude that the others perform satisfactorily with respect to T_{max} they should also be performing better than H2, randomly setting d'_i . The null hypothesis was chosen to be “the difference is greater than or equal to 0”, in other words the heuristic does not lead to a lower T_{max} relative to H2. Null hypothesis was rejected for all heuristics except H2 and H13 (the third column in Table 7) leading us to the conclusion that these two heuristics cannot be shown to be better performers than the random algorithm.

In testing the performance of heuristics with respect to $\overline{AD}$ relative to H1 (the second column in Table 7) the null hypothesis was “difference is greater than or equal to 0”, in other words the heuristics does not lead to a lower $\overline{AD}$ relative to H1. Null hypothesis was rejected for all heuristics, leading us to the conclusion that all heuristics lead to improved stability. The final set of tests were done to test which heuristics lead to a lower $\overline{AD}$ than H2. Thus, the null hypothesis was set to be “difference is greater than or equal to 0” (for the data in the fourth column in Table 7). The null hypothesis was rejected for all heuristics, leading us to the conclusion that these heuristics improved the stability of the schedules better than the random heuristic.

When the results of the tests with respect to the two criteria (T_{max} and $\overline{AD}$) are combined we can conclude that heuristics H9, H10 and H12 lead to not only better stability but also do not lead to any significant increase in maximum tardiness. Furthermore, H10 seems to be the best performer.

Alg	Averages w.r.t. H1		Averages w.r.t. H2	
	$\Delta MT_{1,...}$	$\Delta \overline{AD}_{1,...}$	$\Delta MT_{2,...}$	$\Delta \overline{AD}_{2,...}$
H2	0.8711	-1.6326		
H3	0.5333	-7.2011	-0.3378	-5.5686
H4	0.1044	-4.8088	-0.7667	-3.1762
H5	0.1689	-4.7268	-0.7022	-3.0943
H6	0.1844	-4.8021	-0.6867	-3.1696
H7	0.2889	-6.1095	-0.5822	-4.4769
H8	0.2667	-6.4403	-0.6044	-4.8077
H9	0.0489	-3.2739	-0.8222	-1.6414
H10	0.0822	-5.2609	-0.7889	-3.6283
H11	0.1511	-5.1316	-0.72	-3.4990
H12	0.1711	-4.5262	-0.7	-2.8936
H13	0.7311	-2.5806	-0.14	-0.9481

Table 7 Results

4 Conclusions

In this research, the problem of improving schedule stability (measured by average absolute deviations in start-times) in single-machine rescheduling when a new opera-

tion is inserted into an existing schedule has been addressed. The primary objective in generating a schedule was chosen to be minimizing maximum tardiness.

An algorithm designed for finding solutions which not only are optimal with respect to the T_{max} objective but also have good stability, was used with dummy due-times that are tighter than the actual due-times in order to improve schedule stability further. The initial results are promising. Several of the tested heuristics resulted in improved stability without any statistically significant deterioration in the maximum tardiness objective.

References

1. Allen, J., An interval based representation of temporal knowledge, In *Proceedings of the 7th IJCAI*, Vancouver, Canada, pp.221-226, (1981)
2. Briand, C., La, H. T., Erschler, J., A robust approach for the single machine scheduling problem, *Journal of Scheduling*, 10, pp. 209–221,(2007)
3. Erschler, J., Fontan, G., Merce, C., Roubellat, F., A new dominance concept in scheduling n jobs on a single machine with ready times and due dates, *Operations Research*, 31, pp. 114-127, (1983)
4. Esquirol, P., Lopez, P., Mancel, P., Representation et traitement du temps en ordonnancement (Rapport LAAS No99455) (in French), (1999)

An Iterative Scheme for Parallel Machine Scheduling with Sequence Dependent Family Setups

Liji Shen · Lars Mönch · Udo Buscher

1 Introduction

In this study, we address the parallel machine scheduling problem with setups to minimize the total weighted completion time (TWC). Jobs are grouped into *families* according to their similarities in tooling and operation sequences. In many practical situations, negligible setup is incurred for a job if it belongs to the same family of the preceding job. However, setups are inevitable on each occasion when a machine switches from processing jobs in one family to those in another family [6]. A classical example in this context involves the production of different paint colours on the same machine [4], which also reflects various production realities, especially in chemical, pharmaceutical and food processing industries.

Due to the time-consuming and costly setups, production efficiency can be improved by choosing a long run-length for each family, which leads us to considering batching. More precisely, *batching* refers to the decision of whether or not to schedule similar jobs contiguously [6]. Therefore, a family can be divided into separate *batches*. Jobs belonging to the same batch are to be processed jointly. As a result of integrating batching consideration into the family scheduling model, advantages occur due to the reduced number of setups and therefore, higher machine utilization.

Aside from family setup times, we also take into account *sequence dependencies* (s_{fg}). That is, the length of setup times depends on the family of the current job as well as the family of its immediate predecessor on the same machine. Sequence dependent setup times are commonly deemed as one of the most difficult aspects in scheduling [2]. In addition, we define *initial setup times* (s_{0f}) at the start of the first job on a machine.

Liji Shen
Technische Universität Dresden, Germany
E-mail: liji.shen@tu-dresden.de

Lars Mönch
University of Hagen, Germany
E-mail: lars.moench@fernuni-hagen.de

Udo Buscher
Technische Universität Dresden, Germany
E-mail: buscher@urz.rcs.tu-dresden.de

Furthermore, we focus on the special case of *identical jobs*, i.e., jobs of the same family have identical processing times, which is known to be highly relevant in manufacturing industries [5]. First of all, the repetitive production of the same item represents a typical example. Other applications are encountered, for instance, in the heating process of chemical purification where operations are frequently repeated for the quality promotion of their associated products.

For the specific parallel machine problem $Pm|s\text{-batch}, s_{fg}|TWC$ outlined here, three distinct decisions – batch sizing, batch assignment and batch sequencing – are to be synchronized. However, in order to solve large-sized problem instances with reasonable amount of time, we propose an iterative procedure inspired by [1]. A batch variation procedure is used to determine batch sizes while a variable neighbourhood search algorithm alters batch sequences for improvement. Both of them contribute to assigning batches to machines.

2 An Iterative Heuristic

2.1 List Scheduling Approach

Our list scheduling (BWSPT) heuristic is motivated by the high performance of WSPT dispatching rule while solving the problem $Pm||TWC$. Assume that all batches have the same maximum batch size B_{max} , measures in jobs, and that all batches contain B_{max} jobs, except for the last batch of each family with a smaller number of jobs. We first establish a list by indexing all jobs according to the WSPT-rule. Batches are then formed by adding the next $\min(B_{max}, \tilde{n}_f)$ jobs from the list, where $\tilde{n}_f$ denotes the number of unscheduled jobs of family f .

2.2 Batch Variation Procedure

In order to justify the batch variation procedure (BVP), we first derive some properties and observations regarding batch size variation and the objective value. We conclude that the variation of total weighted completion time is actually a trade-off between the current completion times of two related batches and the aggregate weight of all affected batches. The latter increases with the distance between batches.

Furthermore, the most promising candidates are jobs with smallest and largest weights, according to which, the placements shall also be divided into two groups. Each and every batch of the same family deserves examination after removing a job from its original batch. Consequently, jobs to be considered are the ones with extreme weights in the corresponding batch. They are then added at the end of a batch of the same family.

Regarding batches on different machines, a move concerns not only batch sizing but also the assignment problem, and has higher improvement potential. Accordingly, a greater number of moves are applied. The BVP procedure can be summarized as follows.

- For batches of the same family assigned to the same machine:
 1. Backward-moving a job with smallest weight
 2. Forward-moving a job with largest weight
 3. Removing a job with smallest weight and add it at the end of the machine

- For batches of the same family assigned to different machines:
 1. Moving a job with largest weight and add it at the end of another batch
 2. Moving a job with smallest weight and add it at the end of another machine

Each type of moves is performed repeatedly as long as an improvement occurs during the procedure.

2.3 Batch Scheduling with VNS

The variable neighbourhood search algorithm (VNS) designed for this scheduling problem operates on the final solution representation, i.e. each job is assigned to a batch and each batch is assigned to a certain position on a machine.

The proposed local search method used for VNS consists of two different steps. The first step is balancing the workload of the machines. If the last batch of the machine with the maximum completion time starts later than the completion time of the machine with the smallest workload, the batch is moved to that machine. This step is repeated until no batch can be moved. The second step is the application of a single machine decomposition heuristic proposed by Mehta and Uzsoy for a single machine scheduling problem [3]. From an initial sequence, a small-sized sub-sequence is solved to optimality using complete enumeration. This procedure is repeated until the entire sequence is covered.

Regarding neighbourhood structures, we define the following three different classes:

- *MoveBatch*(x): Select randomly a batch from a machine and remove it. Insert it on a random position on a randomly selected machine. Repeat this step x times.
- *SwapBatch*(x): Select randomly two batches from different machines and exchange their positions. Repeat this step x times.
- *SwapSeq*(x): Randomly select two positions on different machines and exchange the batch sequences starting from that position of at most length x (x or all remaining batches).

Starting with a solution generated using BSPT, metaheuristic VNS is first activated to improve the initial schedule by altering batch sequences. The current solution is then transferred to BVP, which conducts a thorough batch size variation. Afterwards, batch formation is updated and VNS starts anew with improved batch sizes. The entire iterative procedure repeats until a prescribed maximum number of iterations is reached. In this respect, only a small number of iterations is usually required.

3 Computational Results

For the computational experiment we have generated 720 test instances of 10 problem sizes. They range from 20 to 240 jobs, contain $F = 3$ and $F = 6$ families, and are tested on $m \in \{2, 3, 6, 8\}$ parallel machines. Regarding setup times, they are characterized as small $DU[10, 20]$, medium $DU[50, 100]$ and large $DU[100, 200]$, which also satisfy the triangular inequality. In addition, processing times and weights are randomly selected according to $DU[10, 100]$ and $U[0, 1]$, respectively.

To the best of our knowledge, the researched problem has not been addressed so far. We therefore use the results provided by BWSPT as reference for comparison. Additionally, we have developed a mixed integer programming formulation to solve

small instances which consist of at most 10 jobs on 2 to 3 machines. For these cases, the iterative heuristic is able to reach optima instantly.

Table 1 clearly shows the improvement by using BVP and VNS in an isolated manner, as well as the iterative procedure. Note that the results of the first iteration correspond to those obtained by running only VNS. We also observe that most improvement has already been achieved after two iterations. More iterations can then further exploit improvement potential.

Table 1 Improvement of the total weighted completion time

F	n	Improvement of the total weighted completion time (%)					
		BVP	VNS/Iter. 1	Iter. 2	Iter. 3	Iter. 4	final solution
3	20	7,49	8,81	13,69	13,83	13,86	13,86
	30	7,66	11,71	16,24	16,55	16,63	16,64
	40	10,23	11,79	16,55	16,69	16,71	16,72
	60	9,58	12,98	17,46	17,67	17,73	17,75
	80	7,75	11,82	16,75	17,12	17,22	17,24
	90	11,82	13,38	18,36	18,59	18,68	18,71
	120	9,41	13,29	18,08	18,38	18,47	18,50
	160	10,25	13,71	18,74	19,10	19,27	19,32
	180	11,72	12,89	18,70	19,00	19,14	19,18
	240	12,17	12,94	19,17	19,57	19,78	19,84
6	20	8,18	11,38	15,57	15,65	15,65	15,65
	30	10,31	16,00	20,85	21,09	21,12	21,12
	40	10,34	14,48	19,01	19,19	19,21	19,22
	60	13,43	18,49	23,62	23,89	23,97	23,99
	80	9,50	16,28	20,90	21,25	21,34	21,37
	90	14,21	18,81	24,84	25,24	25,37	25,41
	120	10,76	17,76	22,42	22,84	23,02	23,06
	160	11,36	19,19	24,41	25,02	25,28	25,34
	180	13,86	19,28	24,68	25,24	25,52	25,58
	240	13,44	17,89	24,03	24,71	25,07	25,16

References

1. U. Buscher and L. Shen, An integrated tabu search algorithm for the lot streaming problem in job shops, *European Journal of Operational Research*, 199, 395–399 (2009).
2. M. Laguna, A heuristic for production scheduling and inventory control in the presence of sequence-dependent setup times, *IIE Transactions*, 31, 125–134 (1999).
3. S.V. Mehta and R. Uzsoy, Minimizing total tardiness on a batch processing machine with incompatible job families, *IIE Transactions*, 30, 165–178 (1998).
4. C. Monma and C. Potts, On the complexity of scheduling with batch setups, *Operations Research*, 37, 798–804 (1989).
5. G. Mosheiov, D. Oron, and Y. Ritov, Flow-shop batch scheduling with identical Processing-Time Jobs, *Naval Research Logistics*, 51, 783–799 (2004).
6. C.N. Potts and M. Kovalyov, Scheduling with batching: A review, *European Journal of Operational Research*, 120, 228–249 (2000).

Sort & Search exponential-time algorithms for scheduling problems

C. Lenté · M. Liedloff · A. Soukhal · V.
T'kindt

1 Introduction

This communication deals with scheduling theory and exponential-time algorithms. Most of the scheduling problems dealt with in the literature are intractable problems, *i.e.* $\mathcal{NP}$ -hard problems according to complexity theory. Consequently, an optimal solution of such problems can only (unless $\mathcal{P} = \mathcal{NP}$) be achieved by non polynomial time algorithms, the design of which has been the matter of a complete part of the literature over the last decades. Usually, the evaluation of the efficiency of such algorithms is conducted using extensive computational experiments and the challenge is to solve instances of size as high as possible. But, theoretically speaking, several fundamental questions remain open: for exponential-time algorithms can we establish stronger conclusions than their non polynomiality in time? For instance, is it possible to derive upper bounds on their average complexity or their worst-case complexity? This is one task which is usually performed for optimal algorithm solving polynomially solvable problems, since when we provide an optimal polynomial time algorithm we usually also provide information about the number of steps it requires to compute an optimal solution. Why not for $\mathcal{NP}$ -hard problems and exponential time algorithms? This would have the advantage to enable comparisons of such algorithms for a given problem.

C. Lenté
Université François-Rabelais de Tours, Laboratoire d'Informatique, 64 avenue Jean Portalis,
37200 Tours, France
E-mail: christophe.lente@univ-tours.fr

M. Liedloff
LIFO, Université d'Orléans, rue Léonard de Vinci, BP 6759, 45067 Orléans Cedex 2, France
E-mail: mathieu.liedloff@univ-orleans.fr

A. Soukhal
Université François-Rabelais de Tours, Laboratoire d'Informatique, 64 avenue Jean Portalis,
37200 Tours, France
E-mail: ameur.soukhal@univ-tours.fr

V. T'kindt
Université François-Rabelais de Tours, Laboratoire d'Informatique, 64 avenue Jean Portalis,
37200 Tours, France
E-mail: tkindt@univ-tours.fr

But the interest in studying the worst-case time complexity of such algorithms, or even their average complexity, is beyond the simple interest of counting a number of steps. It is related to establishing properties of $\mathcal{NP}$ -hard problems. To illustrate this, assume we deal with a $\mathcal{NP}$ -hard optimisation problem for which a straight enumeration algorithm would calculate an optimal solution requiring $n!$ steps, with n the size of the input. The question is: can this problem admit an exponential time algorithm with a worst-case time complexity lower than that of this enumeration algorithm? Can we solve it using, for instance, at most 2^n steps? Having such a property would not only give a property on the expected difficulty of a problem, but also challenge the design of efficient optimal algorithms: their efficiency should be still evaluated via computational experiments, but they would have also to not exceed the upper bound on the worst-case complexity established on the problem.

The problem of designing exponential-time algorithms with bounds on their worst-case complexity has been the matter of growing interest in combinatorial optimization over the last years ([1, 2]). To the best of our knowledge, no such exponential-time algorithm for scheduling problems can be found in the literature, except one provided in [1] on a single machine problem referred to as $1|prec|\sum C_i$ according to the well-known three-field notation of scheduling problems ([3]). In the literature on exponential-time algorithms, several techniques can be used to derive worst-case bounds ([4]) and in this paper we focus on one of them: the *Sort & Search* method. Noteworthy, we show that this method can be used to derive worst-case complexity results for scheduling problems. The bounds we provide are, obviously, lower than that of obtained by performing a brute-force enumeration of all solutions for these problems. Additionally, we make use of the notation O^* to refer to the worst-case complexity of exponential-time algorithms: for a given exponential-time algorithm with complexity $O(poly(n) \times g(n))$, $poly(n)$ being a polynomial function of the input size n , we refer to $O^*(g(n))$ as its complexity. Therefore, in the notation O^* we suppress the polynomial components which is justified by the exponential growth of $g(n)$.

The remainder is organized as follows. In section 2 we provide the framework of the *Sort&Search* method and in section 3 how it can be applied on some scheduling problems. We provide for each of these problems worst-case time and space complexity bounds.

2 The *Sort & Search* method

A category of techniques for designing exponential algorithms is based on splitting instances at the cost of an increase in the data. In this category, called *Split and List* in [4], an interesting technic is *Sort & Search* which has been first proposed in [5] to solve the discrete knapsack problem in $O^*(2^{\frac{n}{2}})$ time and space. The underlying idea is to create a partition, let's say I_1 and I_2 , of a given instance I . Then, by enumerating all possible partial solutions from I_1 and I_2 we may be able to compute the optimal solution corresponding to the instance I . We illustrate this technic on the discrete knapsack problem defined as follows. Let $O = \{o_1, \dots, o_n\}$ be a set of n objects, each one being defined by a value $v(o_i)$ and a weight $w(o_i)$, $1 \leq i \leq n$. We are also given a positive integer capacity W for the knapsack. The goal is to find a subset $O' \subseteq O$ such that $\sum_{o \in O'} w(o) \leq W$ and $\sum_{o \in O'} v(o)$ is maximum.

The *Sort & Search* technic suggests to partition O into $O_1 = \{o_1, \dots, o_{\lceil n/2 \rceil}\}$ and $O_2 = \{o_{\lceil n/2 \rceil+1}, \dots, o_n\}$. A first table T_1 is built from O_1 by enumerating all subsets $O'_j \subseteq O_1$: a column j of T_1 corresponds to O'_j and is associated with the values $w(O'_j) = \sum_{i \in O'_j} w(o_i)$ and $v(O'_j) = \sum_{i \in O'_j} v(o_i)$. A second table T_2 is built in the same way starting from subset O_2 . These two tables have $2^{\frac{n}{2}}$ columns. Before searching for the optimal solution, we perform a *sort step* on table T_2 : columns j of T_2 are sorted by increasing values of $w(O'_j)$. Besides, for each column in position k after that sorting, we store the index $\ell_k \leq k$ of the column with maximum $v(O'_{\ell_k})$ value *i.e.* $\ell_k = \operatorname{argmax}_{u \leq k} (v(O'_u))$. This processing, which can be achieved by means of a classic sorting procedure, requires $O^*(2^{\frac{n}{2}} \log(2^{\frac{n}{2}})) = O^*(2^{\frac{n}{2}})$ time. Then, a *search step* is applied to find an optimal solution: for each column j of table T_1 , we look for the column k of table T_2 such that $w(O'_j) + w(O'_k) \leq W$ and $v(O'_j) + v(O'_k)$ is maximum. For a given column j , this is achieved by means of a dichotomic search in table T_2 to find column k such that $k = \operatorname{argmax}_{u \in T_2} (w(O'_j) + w(O'_u) \leq W)$. Then, $v(O'_j) + v(O'_{\ell_k})$ is the maximum value of the objectif function when objects of O'_j are put in the knapsack. The examination of all O'_j enables to compute the optimal solution of the problem. The search step can be achieved in $O^*(2^{\frac{n}{2}} \log(2^{\frac{n}{2}})) = O^*(2^{\frac{n}{2}})$ time. Therefore, this *Sort & Search* algorithm requires $O^*(2^{\frac{n}{2}})$ time and space. We provide below a numerical example with $n = 6$ objects, $O = \{a, b, c, d, e, f\}$ and $W = 9$.

O	a	b	c	d	e	f		
v	3	4	2	5	1	3	$O_1 = \{a, b, c\}$	$O_2 = \{d, e, f\}$
w	4	2	1	3	2	5		

T_1	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a, b\}$	$\{a, c\}$	$\{b, c\}$	$\{a, b, c\}$
v	0	3	4	2	7	5	6	9
w	0	4	2	1	6	5	3	7

T_2	$\emptyset$	$\{e\}$	$\{d\}$	$\{f\}$	$\{d, e\}$	$\{e, f\}$	$\{d, f\}$	$\{d, e, f\}$
v	0	1	5	3	6	4	8	9
w	0	2	3	5	5	7	8	10
ℓ	1	2	3	3	5	5	7	8

The table below presents the result of the *search step*: for each column j of T_1 we indicate the column k of T_2 such that $k = \operatorname{argmax}_{u \in T_2} (w(O'_j) + w(O'_u) \leq W)$.

j	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a, b\}$	$\{a, c\}$	$\{b, c\}$	$\{a, b, c\}$
k	$\{d, f\}$	$\{d, e\}$	$\{e, f\}$	$\{d, f\}$	$\{d\}$	$\{d\}$	$\{d, e\}$	$\{e\}$
$w(O'_j) + w(O'_k)$	8	9	9	9	9	8	8	9
$v(O'_j) + v(O'_k)$	8	9	10	10	12	10	12	10

Consequently, the optimal solution value is equal to 12 and can be obtained by putting into the knapsack objects $\{a, b, d\}$.

The *Sort & Search* technic is very powerfull to design exponential algorithms and can be applied to a lot of $\mathcal{NP}$ -hard optimisation problems. Informally speaking, such problems must have the property that two partial solutions can be combined in polynomial time to build a complete solution to the initial instance, and we must be able to set up a sorting step which enables to perform the searching step in no more time than the building of the tables.

3 Application to some scheduling problems

The problems on which the *Sort & Search* method has been applied are displayed in table 1. For each problem, we provide its notation, the worst-case time complexity of the brute-force enumeration and the worst-case time complexity obtained by application of the *Sort & Search* method.

Problem	Enumeration	<i>Sort & Search</i>
$1 d_i \sum_i w_i U_i$	$O^*(2^n)$	$O^*(\sqrt{2}^n)$
$P2 C_{max}$	$O^*(2^n)$	$O^*(\sqrt{2}^n)$
$P2 d_i \sum_i w_i U_i$	$O^*(3^n)$	$O^*(\sqrt{3}^n)$
$P3 C_{max}$	$O^*(3^n)$	$O^*(\sqrt[3]{9}^n)$
$F2 d_i = d, d \text{ unknown}, \sum_i U_i = \epsilon d$	$O^*(2^n)$	$O^*(\sqrt{2}^n)$

Table 1 Bounds on the worst-case time complexity for some NP-hard scheduling problems

In this abstract, we only provide details about the application of the *Sort & Search* method to the $1|d_i|\sum_i w_i U_i$ problem (proofs are omitted). Consider that each job i is defined by a processing p_i , a due date d_i and a tardiness penalty w_i . The aim is to compute an optimal schedule s which minimizes the weighted number of late jobs defined by $\sum_{i=1}^n w_i U_i$ with $U_i = 1$ if $C_i(s) > d_i$ and $U_i = 0$, otherwise. This problem, which is referred to as $1|d_i|\sum_i w_i U_i$, has been shown $\mathcal{NP}$ -hard in the weak sense ([6] and [7]). Without loss of generality we assume that jobs are numbered such that $d_1 \leq d_2 \leq \dots \leq d_n$. We first establish some properties.

Theorem 1 *Let E be a set of early jobs, i.e. $\forall i \in E, C_i \leq d_i$. Either there is no feasible schedule s in which all jobs in E are early, either there exists an optimal schedule in which all jobs in E are sequenced by increasing value of their due date d_i (Earliest Due Date rule, EDD).*

Theorem 2 *Let s_{EDD} be the schedule obtained by the EDD rule on a set of early jobs E , with $L_{max}(s_{EDD}) \leq 0$. There exists a feasible schedule of all jobs in E starting at time t iff $L_{max}(s_{EDD}) + t \leq 0$.*

First, consider the *Enum* algorithm which solves the problem by a brute-force enumeration of all schedules. From theorem 1 we can evince that *Enum* has only to enumerate all the sets of possible early jobs: for each set E of early jobs, a complete schedule s is obtained by sequencing them using EDD rule and adding at the end the tardy jobs. By keeping the schedule s with the minimal value of $\sum_i w_i U_i$, *Enum* can solve optimally the problem. As there are 2^n sets of possible early jobs, *Enum* has a worst-case time complexity in $O^*(2^n)$ time.

We make use of the *Sort & Search* method to derive the following optimal algorithm, referred to as *SorSea*. Let be $I_1 = \{1, \dots, \lfloor \frac{n}{2} \rfloor\}$ and $I_2 = \{\lfloor \frac{n}{2} \rfloor + 1, \dots, n\}$ a decomposition into two jobs sets of the initial instance to solve. Starting from set I_1 , algorithm *SorSea* builds a sequence of early jobs scheduled first, whilst starting from set I_2 it builds a sequence of early jobs scheduled right after the early jobs of I_1 . Let $s_1^j \subseteq I_1$ (resp. $s_2^k \subseteq I_2$) be a sequence of early jobs sorted by the EDD rule, and $\bar{s}_1^j = I_1 - s_1^j$ (resp.

$\bar{s}_2^k = I_2 - s_2^k$) be the sequence of tardy jobs (in any order). The decomposition of a schedule s computed by *SorSea* is presented in figure 1. We also define $P(A) = \sum_{i \in A} p_i$. We have:

$$\sum_{i=1}^n w_i U_i(s) = \sum_{i \in \bar{s}_1^j} w_i + \sum_{i \in \bar{s}_2^k} w_i.$$

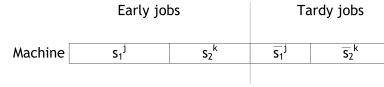


Fig. 1 Decomposition of a schedule s for the $1|d_i| \sum_i w_i U_i$ problem

SorSea algorithm builds a table T_1 in which each column j is associated with a sequence $s_1^j \subseteq I_1$ of at most $\frac{n}{2}$ jobs. Therefore, table T_1 contains at most $2^{\frac{n}{2}}$ columns. For each column j we store the values $P(s_1^j)$ and $\sum_{i \in \bar{s}_1^j} w_i$. *SorSea* algorithm also builds a table T_2 in which column k is associated with a sequence $s_2^k \subseteq I_2$ of at most $\frac{n}{2}$ jobs, leading also to at most $2^{\frac{n}{2}}$ columns. However, in table T_2 they are sorted by decreasing values of $L_{max}(s_2^k)$. For each column k we store the values $L_{max}(s_2^k)$, $\sum_{i \in \bar{s}_2^k} w_i$ and $wU_{min}(s_2^k) = \min_{\ell \geq k} (\sum_{i \in \bar{s}_2^\ell} w_i)$. For a given column j of T_1 , i.e. with associated partial sequences s_1^j and $\bar{s}_1^j$, *SorSea* searches in $O^*(\frac{n}{2})$ time in T_2 the column k such that:

$$k = \operatorname{argmin}(u \in T_2 / P(s_1^j) + L_{max}(s_2^u) \leq 0).$$

By making use of theorem 2 we can deduce that all columns $\ell \geq k$ in table T_2 correspond to all the partial schedules s_2^ℓ with no tardy job if they are scheduled after s_1^j . The value of the smallest $\sum_i w_i U_i(s)$ value in a schedule s starting by the partial sequence s_1^j of early jobs and with jobs in $\bar{s}_1^j$ tardy is then given by:

$$\sum_i w_i U_i(s) = \sum_{i \in \bar{s}_1^j} w_i U_i + wU_{min}(s_2^k).$$

Therefore, by computing for each column j of T_1 the above value, *SorSea* algorithm can compute the optimal solution of the $1|d_i| \sum_i w_i U_i$ problem.

Theorem 3 *SorSea* algorithm solves the $1|d_i| \sum_i w_i U_i$ problem with a worst-case time and space complexity in $O^*(2^{\frac{n}{2}}) = O^*(\sqrt{2}^n) \approx O^*(1.4142^n)$.

The *Sort & Search* method can also be applied in a similar way to the other scheduling problems quoted in table 1. This will be illustrated during the presentation at the conference.

References

1. Woeginger, G. Exact algorithms for NP-hard problems: A survey. M. Junger, G. Reinelt, G. Rinaldi, eds., *Combinatorial Optimization – Eureka, You Shrink!*. Springer, vol 2570, 185–207 (2003).
2. Woeginger, G. Space and Time Complexity of Exact Algorithms: Some Open Problems. R. Downey, M. Fellows, F. Dehne, eds., *Parameterized and Exact Computation*. Springer, vol 3162, 281–290 (2004).

3. Graham, R. L., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* **5** 287–326 (1979).
4. Fomin, F., Kratsch, D. Exact Exponential Algorithms. *Springer* (2010).
5. E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21:277–292 (1974).
6. Karp, R.M. Reducibility among combinatorial problems. *Complexity of Computer Computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972)* 85–103 (1972).
7. Lawler, E.L., J.M. Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science* **16** 77–84 (1969).

Hybridizing local search with MIP for unrelated parallel machine scheduling problems with makespan minimization

Krzysztof Fleszar · Christoforos Charalambous

This work proposes a local search method hybridized with mixed-integer programming (MIP) for two *unrelated parallel machine scheduling problems with makespan minimization*. It shows that such a hybridization allows for finding very good solutions in short processing times, despite repeatedly solving MIP subproblems.

The first considered problem, the *unrelated parallel machine scheduling* (UPMS), is that of scheduling without preemption a set of jobs, each available at time zero, on a set of unrelated machines. The machines are unrelated in that the processing time of a job depends on the machine on which it is processed. The term *span* is used to denote the completion time of the last job on a machine. The objective is to minimize the *makespan*, which is defined as the maximum span over all machines.

The second considered problem, the *UPMS with sequence-dependent and machine-dependent setup times* (UPMSwS), differs from UPMS in that a setup must be performed before processing each job, with setup times that are sequence-dependent (depend on which job was processed before) and machine-dependent (depend on the machine on which the job is processed).

Several works have been published on the UPMS without setups. The most recent are two papers of Peyro and Ruiz [3,4], who indicate that the state-of-the-art methods for the UPMS are Partial [8], Recovering Beam Search (RBS) [5], DIG [3], and 2JDi(90) [4]. Very few works have addressed the UPMSwS. Only four methods for solving UPMSwS have been proposed so far: Partitioning Heuristic [1], Metaheuristic for Randomized Priority Search (Meta-RaPS) [9], Tabu Search [7], and Ant Colony Optimization (ACO) [2]. The latter work indicates that the state-of-the-art methods for the UPMSwS are Meta-RaPS and ACO.

We propose a hybrid local search method for solving both UPMS and UPMSwS. First, an initial solution is generated. Then, the solution is improved using the *first*

Krzysztof Fleszar
American University of Beirut (AUB), Olayan School of Business
P.O. Box 11-0236, Riad El Solh, Beirut 1107 2020, Lebanon
E-mail: kfleszar@gmail.com, E-mail: kf09@aub.edu.lb

Christoforos Charalambous
Frederick University, Department of Computer Science and Engineering
P.O. Box 24729, 1303 Nicosia, Cyprus
E-mail: c.charalambous@frederick.ac.cy

local search, which is based on *simple moves*. When this local search reaches a local optimum, the *second local search* is initiated, which attempts to determine an improving *compound move* by solving a MIP problem. If this attempt is successful, the determined compound move is adopted and the first local search is restarted. This process is repeated until the second local search fails to improve the solution.

The first local search is based on two types of *simple moves*: a *transfer* of a job from its current machine to another machine and a *swap* of two jobs residing on different machines. The second local search is based on *compound moves*, which involve any number of transfers as long as at most one job is removed from each machine and at most one job is added to each machine. An improving compound move is determined by solving exactly a MIP model. The solution of the MIP is fast, because the space of improving compound moves is reduced by the earlier application of the first local search.

Both local search methods aim at minimizing makespan. Additionally, an auxiliary objective minimizing the sum of spans is used. When two solutions are compared, the solution with the smaller makespan is preferred, but if both solutions have the same makespan, the solution with the smaller sum of spans is preferred. Within the MIP used in the second local search, this behavior is achieved by minimizing the makespan plus a small weight times the sum of spans.

Since the two problems considered here are different, different approaches are adopted in some parts of the proposed method. A solution of UPMS is represented by an assignment of jobs to machines, because spans depend only on which jobs are assigned to machines and not on the order of jobs. In contrast, a solution of UPMSwS must also include the order of jobs on each machine, because spans depend also on sequence-dependent setup times.

Initial solutions in both problems are also generated differently. For UPMS, initial solutions are generated based on the linear relaxation. For UPMSwS, initial solutions are generated by heuristically adding jobs to machines in a random order.

A major difference between the algorithms proposed for UPMS and UPMSwS is in the evaluation of solutions resulting from adopting simple or compound moves. First note that all simple and compound moves can be decomposed into a set of elementary modifications, one modification for each affected machine. The types of elementary modifications are: *addition* (one job is added to the machine and nothing is removed), *removal* (one job is removed from the machine and nothing is added), and *exchange* (one job is removed from the machine and another is added). Evaluation of any simple or compound move is equivalent to evaluating an appropriate set of elementary modifications. This evaluation is trivial in UPMS: spans are increased (decreased) by processing times of added (removed) jobs. In UPMSwS, spans depend additionally on sequences of jobs on machines, so when adding a job to a machine, its position in the sequence must be decided. Additionally, it is often beneficial to reoptimize the sequence of jobs on a machine after adding and/or removing a job.

Finding a minimum-span sequence of jobs on a machine in the UPMSwS problem is equivalent to solving the *asymmetric traveling salesman problem* [6]. *Contract-or-patch* (COP) heuristic [6] is used within our algorithm to construct good solutions for this problem. Additionally, a *sequence local search* is used, which repeatedly performs the best-improving relocation of a single job within the sequence until no improvement is possible. These two heuristics are used every time an elementary modification is considered during the first and second local search for UPMSwS.

Our algorithm for UPMS was compared to the state-of-the-art methods on the benchmark problem instances of Fanjul-Peyro and Ruiz [3], which are divided into seven classes defined by the distributions from which job processing times were generated. Our algorithm performs better than Partial [8], RBS [5], and DIG [3] on all problem classes and better than 2JDi(90) [4] on all but one class, which allows for a job to have a processing time on one machine 100 times bigger than on another. Excluding this class (which may be seen as unrealistic) the average deviation (from best-known solutions) of our algorithm is 0.04% compared to 1.05% for Partial, 0.95% for RBS, 0.36% for DIG, and 0.13% for 2JDi(90). If this class is included, the average deviation of our algorithm is 0.29% compared to 1.31% for Partial, 1.11% for RBS, 0.50% for DIG, and 0.17% for 2JDi(90). In terms of computation time, our algorithm is on average at least 3.5 times faster than any of the other methods, taking into account different processors used.

Our algorithm for UPMSwS was tested on the benchmark problem instances available from <http://www.schedulingresearch.com> and was found to perform much better than the two state-of-the-art methods: Meta-RaPS [9] and ACO [2]. Average deviations (from best-known lower bounds) are 6.35% for our algorithm compared to 8.02% for MetaRaPS and 8.85% for ACO. Our algorithm finds solutions better than those of MetaRaPS for 1454 out of 1620 benchmark problem instances (worse for only 16) and better than those of ACO for 1574 instances (worse for only 7). In terms of computation time, our algorithm is 1-2 orders of magnitude faster than ACO (computation times of Meta-RaPS are not available).

Acknowledgements This research was supported by the Cyprus Research Promotion Foundation (Project TEXNO/0603/08 - NETEXNOPRO).

References

1. Al-Salem, A.: Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar* **17**, 177–187 (2004)
2. Arnaout, J., Rabadi, G., Musa, R.: A two-stage Ant Colony Optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing* **21**(6), 693–701 (2010)
3. Fanjul-Peyro, L., Ruiz, R.: Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research* **207**, 55–69 (2010)
4. Fanjul-Peyro, L., Ruiz, R.: Size-reduction heuristics for the unrelated parallel machines scheduling problem. *Computers & Operations Research* **38**, 301–309 (2011)
5. Ghirardi, M., Potts, C.N.: Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research* **165**(2), 457–467 (2005)
6. Glover, F., Gutin, G., Yeo, A., Zverovich, A.: Construction heuristics for the asymmetric TSP. *European Journal of Operational Research* **129**, 555–568 (2001)
7. Helal, M., Rabadi, G., Al-Salem, A.: A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research* **3**(3), 182–192 (2006)
8. Mokotoff, E., Jimeno, J.L.: Heuristics based on partial enumeration for the unrelated parallel processor scheduling problem. *Annals of Operations Research* **117**(1–4), 133–150 (2002)
9. Rabadi, G., Moraga, R.J., Al-Salem, A.: Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing* **17**, 85–97 (2006)

Flow Shop Scheduling on GPU

Tomáš Zajíček · Přemysl Šůcha

1 Introduction

GPGPU (General Purpose computing on Graphics Processing Units) or GPU computing are terms denoting general purpose scientific and engineering computing on GPU (graphics processing unit). Existing GPU applications are mostly oriented on fields such as medical imaging and natural resource exploration, and creating breakthrough applications in areas such as image recognition and real-time HD video playback and encoding. On the other hand, in the recent time there is a growing attempt to use this specialized hardware to solve NP-hard combinatorial problems.

Nowadays, majority of scientific GPGPU use CUDA (Compute Unified Device Architecture) [1]. It is a parallel computing architecture allowing to write programs for the CUDA GPU, in the C language, independently on the number of available GPU computational cores.

Basically, there are two GPGPU computing models on CUDA. In a *homogeneous computing model* all computations are performed on GPU. On the contrary, in a *heterogeneous computing model* the sequential part of the application runs on the CPU and the computationally-intensive part is accelerated by the GPU. From the user's perspective, the application just runs faster because it is using the high-performance of the GPU to boost performance. The weak point of the heterogeneous computing model is the communication bandwidth between CPU and GPU which often constitute a bottleneck in many applications.

In recent years there are several works proposing solutions of NP-hard combinatorial problems on GPUs. Almost all works use the heterogeneous computing model. One of the earliest works is [2] which proposes a parallel tabu search algorithm for the traveling salesman problem and for the flow shop problem. This approach is implemented using shader programming instead of using CUDA framework. GPGPU approach to the quadratic 3-dimensional assignment problem is suggested in [3]. Identically as in the previous paper they use tabu search algorithm which iteratively use GPU for the neighbourhood local search. A GPU based parallelization of a metaheuristic algorithm for the flexible job shop scheduling problem is presented in [4]. The authors propose two double-level parallel meta-heuristic algorithms based on their method of the neigh-

Tomáš Zajíček
Department of Control Engineering, Faculty of Electrical Engineering
E-mail: zajictom@fel.cvut.cz

Přemysl Šůcha
Department of Control Engineering, Faculty of Electrical Engineering
E-mail: suchap@fel.cvut.cz

bourhood determination. Another method using the heterogeneous computing model for the knapsack problem is presented in [5]. The proposed method is based on the dense dynamic programming. A genetic algorithm implemented in the homogeneous computing model is presented in [6]. It is based on island models and the solution is examined using Griewank's, Michalewicz's and Rosenbrock's artificial benchmark functions.

This paper presents a genetic algorithm (GA) implemented on a GPU for permutation flow shop scheduling problem [7]. This approach is based on the homogeneous computing model in order to maximize the algorithm efficiency. The genetic algorithm is based on work presented in [6]. In addition, we suggest several improvements (e.g. another mechanism for individuals migration) allowing to apply this algorithm on a wider group of NP-hard combinatorial problems. Our experimental results show significant speedup with respect to the sequential algorithm version.

2 Problem Statement

There are n jobs $i = 1, \dots, n$. Each job is ready for processing at time zero and it has to be processed on machines $M_1, M_2, \dots, M_m$ in this order where m is the number of machines. Machines can only process one job at a time and jobs can be processed by only one machine at a time. Processing of job i on machine M_j is denoted as operation $O_{i,j}$. The processing time of operation $O_{i,j}$ is equal to $p_{i,j} > 0$. Thus the problem is to find the order π_i of jobs so that the completion time of the last operation is minimal, i.e. the objective is to minimize schedule length $C_{max} = \max_{i=1, \dots, n} \{s_{i,m} + p_{i,m}\}$. Moreover, the order in which jobs are processed is the same for each machine. Using the standard three-field notation, the problem can be denoted as $F||C_{max}$ [7].

3 GPU-Based Approach Outline

The presented genetic algorithm uses the islands based model. Each island is constituted by a group of individuals forming one local population. This parallel model is mapped on the CUDA architecture so that each CUDA thread process one individual while the island corresponds to a CUDA block of threads. The migration between islands uses migration buffer in global memory. This buffer is divided into several blocks. Access into these blocks is controlled by buffer locks stored in the global memory. These locks are accessed by atomic read-modify-write operation only, i.e. `atomicExch`. This approach avoids mutation of individuals during the migration caused by simultaneous access of two different threads into the same global memory address. This fact was not solved in [6] since the solved problem was not sensitive to this unpredictable mutation.

Each individual is processed by one thread while all data of the thread are stored in shared and local memory. Each individual is represented by vector of π_i , i.e. order of jobs. Matrix of processing times $p_{i,j}$ is stored in textures memory which is optimized for 2D access and is cached by each GPU multiprocessor. In the thread initialization, a randomly generated individual is loaded from the global memory. Consecutively, the algorithm iterates through the fitness function (C_{max}) evaluation, migration, tournament selection, single point crossover and mutation. The mutation operator is realized by a position exchanging of two jobs. The thread iterates until the stop condition is

	n/m	Blocks/ Threads	Time on CPU [s]	Time on GPU [s]	Speedup	Initial C_{max}	C_{max} on CPU	C_{max} on GPU
ta001	20/5	64/256	404.3	1.2	336.9	1517	1297	1278
ta011	20/10	64/256	327.0	2.0	163.5	2057	1586	1595
ta021	20/20	64/256	490.4	3.8	129.1	2789	2316	2315
ta031	50/5	64/128	280.5	3.5	80.1	3222	2724	2729
ta041	50/10	64/128	257.9	4.8	53.7	3745	3140	3089
ta051	50/20	64/128	356.7	8.5	42.0	4764	3961	3973
ta061	100/5	64/64	299.6	8.9	33.7	6037	5495	5495
ta071	100/10	64/64	380.5	10.8	35.2	6824	5812	5814
ta081	100/20	64/64	467.2	17.5	26.7	7781	6480	6530
ta091	200/10	64/32	416.4	30.2	13.8	12097	10957	10991
ta101	200/20	64/32	439.4	41.5	10.6	13353	11617	11660

Table 1 The comparison of the GPU implementation against the CPU implementation.

met, i.e. the specified number of processed iterations is reached. Finally, each thread writes its result into global memory.

4 Results and Conclusions

For the measurement of the programs performance a PC with Windows Server 2008 R2 Enterprise OS, 8 [GB] RAM, AMD Phenom II X4 945 3.0 [GHz] CPU and an NVIDIA Tesla C1060 GPU with 240 CUDA cores has been used. Two equivalent implementations of the same genetic algorithm for solving the permutation flow shop scheduling problem have been implemented. The first one on CPU and the second one on GPU. Both programs have been compared on benchmark data from [8]. The number of iterations performed by GA was set to 1000 in both algorithm versions in this experiment.

The results of the program runs are showed in Table 1. The problem size is defined by the count of jobs n and the count of machines m . The number of the blocks of threads (i.e. islands) was set to 64. The number of the threads per a block was ranged between 32 and 256 (indirectly proportional to the memory requirements of the problem - jobs count). Column Initial C_{max} in Table 1 represents the best C_{max} in the initial population while two consequent columns contain resulting C_{max} after 1000 algorithms iterations.

The results of the measurements show that both implementations of the same GA came to the similar results but the speed of the GPU implementation is two to three orders of magnitude higher than the CPU implementation speed (the speedup is from 11 to 337). The acceleration of GPU implementation against CPU implementation is directly proportional to the number of threads that were running in the block.

References

1. NVIDIA, . NVIDIA CUDA C Programming Guide 3.1. NVIDIA; 2010.
2. Janiak, A., Janiak, W., Lichtenstein, M.. Tabu search on gpu. Journal of Universal Computer Science 2008;14(14):2416–2427.

3. Luong, T.V., Loukil, L., Melab, N., Talbi, E.G.. A gpu-based iterated tabu search for solving the quadratic 3-dimensional assignment problem. *Computer Systems and Applications, ACS/IEEE International Conference on* 2010;0:1–8.
4. Boejko, W., Uchroński, M., Wodecki, M.. Parallel hybrid metaheuristics for the flexible job shop problem. *Computers and Industrial Engineering* 2010;59:323–333.
5. Boyer, V., Baz, D.E., Elkihel, M.. Dense dynamic programming on multi gpu. Technical Report LASS-CNRS no. 10509; 2010.
6. Pospichal, P., Jaros, J., Schwarz, J.. Parallel genetic algorithm on the cuda architecture. In: *Applications of Evolutionary Computation*; vol. 6024 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg; 2010, p. 442–451.
7. Brucker, P.. *Scheduling Algorithms*. SpringerVerlag; 5th ed.; 2007. ISBN 354069515X.
8. Taillard, E.. *Benchmarks for basic scheduling problems*. 1989.

Computational Comparison of Neighbourhoods for the Job Shop Scheduling Problem with Total Weighted Tardiness Objective

Jens Kuhpfahl · Christian Bierwirth

1 Introduction

Minimization of the makespan is the predominant objective considered for Job Shop Scheduling Problems (JSSP). If, however, job specific due dates must be respected (e.g. to avoid penalties for missing contracted delivery dates), due date objectives, such as the minimization of the total weighted tardiness, become important.

The literature offers relatively few results for JSSP with tardiness objectives. Besides an exact method [10] and a shifting bottleneck heuristic [9] there are some results based on metaheuristic concepts combined with local search methods (see e.g. [2],[3],[5] and [6]). This work presents a computational comparison of existing and new neighbourhoods for JSSP with total weighted tardiness objective. Thereby we aim at identifying the most suitable neighbourhood definitions for metaheuristics based on local search.

2 Problem Definition and Model

The JSSP is defined as follows: a set of n jobs has to be processed on a set of m machines. Each job consists of m operations that have to be processed in a given sequence on the machines. Preemption in the processing of an operation is not allowed. Every machine can perform at most one job at a time. Every job can start earliest at ready time r_j and should be fulfilled at its due date d_j . The total weighted tardiness is defined by: $\sum T = \sum_{j=1}^n w_j \max\{0, C_j - d_j\}$, where w_j is the weight attached to job j and C_j is its completion time in a schedule.

The disjunctive graph representation for the JSSP (see Adams et al. [1]) can be extended to the consideration of a tardiness objective by the introduction of a sink W_j for every job j . An example with 3 jobs and 3 machines is shown in Fig. 1. The possible precedences of jobs on the machines are represented by pairs of disjunctive arcs (a pair

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

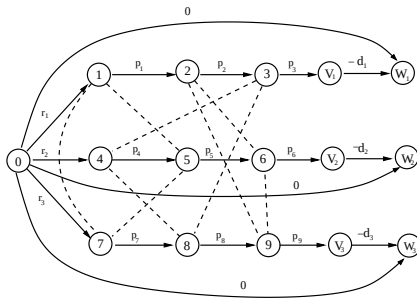


Fig. 1 Disjunctive Graph G
(dashed line = pair of disjunctive arcs).

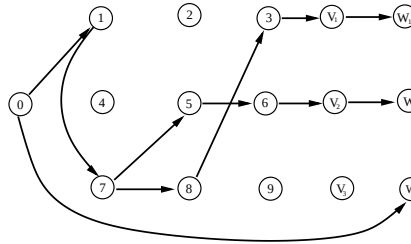


Fig. 2 Critical Tree CT .

of disjunctive arcs is depicted by one dashed line in Fig. 1). A solution corresponds to a selection of one arc from every pair of the disjunctive arcs, such that the resulting directed graph G' is acyclic.

Each of the n vertices W_j is connected with the starting vertex 0 via a longest paths. By superposing these n paths, a critical tree CT results with root 0 and exactly n leaves (see Fig. 2). We define the length $S(CT)$ of this tree as the weighted sum of the lengths of all longest paths. $S(CT)$ corresponds to the total weighted tardiness of the underlying schedule.

Every directed arc in CT , that describes a precedence relation on a machine, can be reverted to generate an alternative feasible solution. A maximal connected sequence of these "critical arcs" on a longest path, and thus in CT , is called a critical block [8]. Critical blocks are the basic structures in neighbourhood definitions for local search.

3 Neighbourhoods for Job Shop Scheduling

The following neighbourhoods have been proposed for the JSSP with makespan objective and can be easily adapted to the critical tree. The critical adjacent pairwise interchange (CRAPI) of two operations incident to a critical arc is introduced by van Laarhoven et al. [11]. Nowicki and Smutnicki [8] reduce the neighbourhood (RCRAPI) by considering only start and end operations of critical blocks since changes of arcs within a block cannot yield better solutions. Matsuo et al. [7] extend the neighbourhood (3CRAPI) by interchanging additional arcs related to predecessors or successors in the machine sequence of critical operations to obtain a significant change of the longest path. The permutation of three critical operations (3CRAP) and the critical shift (CRSHIFT) of an operation within a critical block are proposed by Dell'Amico and Trubian [4]. The CRSHIFT operator performs a maximal shifting that yields a feasible solution.

4 New Neighbourhood Definitions

We introduce six new neighbourhoods, which extend and supplement the concepts proposed in the literature. In 2RCRAPI an interchange of the arcs related to the start and end operation in a critical block is made in one step. The ICRAPI neighbourhood inverts every second arc in a critical block beginning with the arc of the start operation.

Both neighbourhoods aim at breaking block structures in the *CT*. The extension of 3CRAP to 4CRAP by performing the permutation of four operations is expected to yield a positive effect for large instances. The neighbourhoods DCRSHIFT and RDCRS SHIFT extend the shifting idea. In DCRSHIFT, the first operation of a selected arc is shifted to the end of the corresponding critical block and the second operation is shifted to the start of the block. RDCRS SHIFT performs these shifts in reverse direction. The start operation of the considered critical block is shifted in front of the second operation of a selected arc, the end operation is shifted behind the first operation of the same arc. Last but not least, we define the TRCRSEQ neighbourhood where a sequence of operations within a critical block is inverted.

5 Our Presentation

In our talk, we outline the neighbourhoods known from the literature and derive new neighbourhoods in detail. We also present a performance comparison of the neighbourhoods. For this purpose, we have chosen ten instances from OR-Library: LA16-LA20 (each with 10 jobs and 10 machines) and LA31-LA35 (each with 30 jobs and 10 machines) [12]. To generate due dates and weights, we applied the procedure of [6] with two different due date factors f . Each instance was analyzed with a tight factor $f = 1.3$ and a larger factor $f = 1.5$. Starting with 1000 random solutions for every setting, the performance of every neighbourhood has been tested by iterative improvement until a local optimum was obtained.

By evaluating average improvement rates and further performance indicators, we want to identify an appropriate neighbourhood for metaheuristics based on local search. Moreover, we analyze the heterogeneity of the obtained local optima for identifying the useful neighbourhoods that can efficiently combined.

References

1. J. Adams, E. Balas, D. Zawack, The Shifting Bottleneck Procedure for Job Shop Scheduling, *Management Science* 34(3), 391-401 (1988)
2. V.A. Armentano, C.R. Scrich, Tabu search for minimizing total tardiness in a job shop, *International Journal of Production Economics* 63(2), 131-140 (2000)
3. K. Bülbül, A Hybrid Shifting Bottleneck-Tabu Search Heuristic for the Job Shop Total Weighted Tardiness Problem, *Computers & Operations Research* 38 (6), 967-983 (2011)
4. M. Dell'Amico, M. Trubian, Applying tabu search to the job-shop scheduling problem, *Annals of Operations Research* 41(3), 231-252 (1993)
5. I. Essafi, Y. Mati, S. Dauzère-Pérès, A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem, *Computers & Operations Research* 35(8), 2599-2616 (2008)
6. S. Kreipl, A large step random walk for minimizing total weighted tardiness in a job shop, *Journal of Scheduling* 3(3), 125-138, (2000)
7. H. Matsuo, C.J. Suh, R.S. Sullivan, A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem, Working paper 03-04-88, University Texas (1988)
8. E. Nowicki, C. Smutnicki, A Fast Taboo Search Algorithm for the Job Shop Problem, *Management Science* 42(6), 797-813 (1996)
9. M. Pinedo, M. Singer, A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop, *Naval Research Logistics*, 46(3), 1-17 (1999)
10. M. Singer, M. Pinedo, A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops, *IIE Transactions* 30(2), 109-118 (1998)
11. P.J.M. van Laarhoven, E.H.L. Aarts, J.K. Lenstra, Job Shop Scheduling by Simulated Annealing, *Operations Research* 40(1), 113-125 (1992)
12. OR-Library. <http://people.brunel.ac.uk/~mastjjb/jeb/info.html>

Minimizing changeover times of a make-and-pack production process: model and case study

Philipp Baumann · Norbert Trautmann

1 Introduction

A make-and-pack production facility is a two-stage production facility where the two stages are connected by a set of storage tanks with limited capacities. On the make stage, various types of products are manufactured and transferred into storage tanks. On the pack stage, the products are extracted from the tanks and packed in several package types. This type of production facility is used in many process industries to produce, e.g. food and beverages, hair dyes, personal hygiene products and toothpaste.

We consider a case study referring to a real-world make-and-pack production process at The Procter & Gamble Company (cf. [5]). The planning problem consists in computing a production schedule such that a given demand of packed products is fulfilled within a prescribed planning horizon, all technological constraints are satisfied, and the total time required for changeovers is minimized. Here, minimizing changeover times increases the degree of resource utilization and decreases the amount of energy and polluting materials required for production.

To the best of our knowledge, only [2] present an optimization model which covers all constraints occurring in this case study. However, this model is tailored to makespan minimization and is not applicable to minimization of changeover times. Other related models known from the literature (cf. e.g., [1] or [3]) capture only some, but not all characteristics of the case study considered here.

In the present paper, we develop an MILP formulation for computing production schedules with minimal total changeover times for such make-and-pack production process. We show that a substantial reduction in CPU time can be achieved by introducing various types of symmetry-breaking constraints. For our computational analysis, we have generated a set of problem instances by varying the demand data of the case study. The results of the analysis show that moderate-sized instances can be solved to optimality within short CPU times.

Philipp Baumann · Norbert Trautmann
University of Bern, Department of Business Administration, 3012 Bern, Switzerland
E-mail: philipp.baumann@pqm.unibe.ch, norbert.trautmann@pqm.unibe.ch

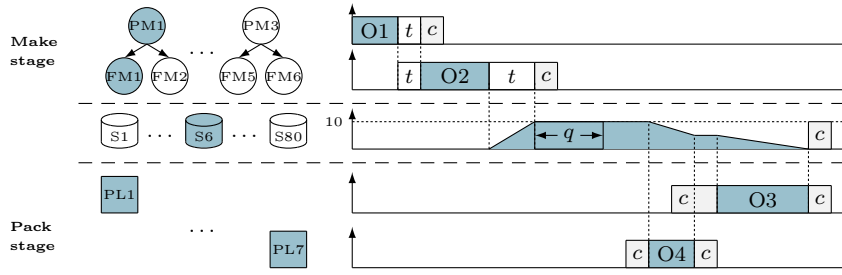


Fig. 1 Production and packing processes

2 Planning problem

The production and packing process described in the case study of [5] is depicted in Fig. 1. The make stage consists of three groups of premix and final-mix units which operate batches of fixed size. Both make operations (O1 and O2) have to be executed in immediate succession, i.e., the final-mix operation (O2) must start as soon as the output of the premix operation has fully been transferred into the final-mix unit. Material transfers require non-negligible amounts of time (t) during which both units involved are occupied. After production, a batch may be split, and it needs to be held in one or two storage tanks for at least a given quarantine time (q). In total, 80 storage tanks decouple the make stage and the pack stage. It is not allowed to store different batches concurrently in the same tank. The pack stage consists of seven packing lines with continuous material flows. The duration of the packing operations (O3, O4) depends on the processing unit. Processing units need to be washed-out between the processing of products belonging to different wash-out families. Moreover, each package type requires a specific setup of the packing line. In Fig. 1, wash-out and setup times are denoted with c .

The planning problem consists in assigning a processing unit and a start time to each operation such that a given demand of packed products is met within a prescribed planning horizon, all technological constraints are satisfied, and the total time for changeovers, i.e. wash-outs and package-type setups, is minimized. The due date for all packed products corresponds to the end of the planning horizon.

3 Model formulation

The main decision variables of the MILP formulation refer to sequences of batches, i.e., we use a continuous representation of time. In total, we use five groups of variables for modelling the following decisions:

- start and finish time of each operation (continuous variables)
- flow of material from the make stage to the pack stage through the storage tanks (continuous variables and binary variables)
- allocation of the operations to the processing units, and sequencing of the operations on the individual processing units (binary variables representing unit-specific immediate precedence relationships, cf. [4])
- allocation of the operations to the storage tanks (binary variables)
- sequencing of the operations in the individual storage tanks (binary variables representing general precedence relationships, cf. [6])

The total changeover times are computed straightforwardly using the variables of group (c). Moreover, all technological characteristics of the production process (cf. Section 2) are formulated as constraints of the MILP using the variables of groups (a)–(e).

Without loss of generality, we add two groups of symmetry-breaking constraints in order to shrink the solution space. The first group refers to identical batches, i.e., batches that can be interchanged in all feasible schedules without affecting the objective function value. We impose an arbitrary sequence of these batches by adding constraints that refer to the variables of group (a). The second group of constraints refers to storage tanks of identical capacity. We reduce the number of possible allocations of batches to these tanks by adding constraints that refer to the variables of group (d).

4 Computational results

For the numerical evaluation of our model, we used the 9 demand vectors presented in [2]. In the same paper, an upper bound *ub* on the minimal makespan for each of these demand vectors has been computed. With respect to the length of the planning horizon *H*, we consider 4 scenarios for each demand vector by increasing *ub* by 0%, 10%, 20%, and 30%. Correspondingly, our test set consists of 36 problem instances. The results, which we obtained by Gurobi 4.0 on a standard PC, can be summarized as follows:

1. 30 out of the 36 instances could be solved to optimality. The time limit of 5 hours was reached 6 times, 5 times without finding a feasible solution.
2. A substantial reduction of up to 62% in changeover times can be achieved by enlarging the planning horizon by 10%. Enlarging the planning horizon usually accelerates the search for an optimal solution.
3. When the model is solved without symmetry-breaking constraints, the number of instances for which the time limit is reached increases to 14 instances. For those instances that are still solved to optimality, the total CPU time required increases by 683%.

Acknowledgements This work was supported by the Swiss National Science Foundation, grant 205121-125106.

References

1. C. Artigues and D. Feillet. A branch and bound method for the job-shop problem with sequence-dependent setup times. *Ann. Oper. Res.*, 159:135–159, 2009.
2. P. Baumann and N. Trautmann. An MILP approach to short-term scheduling of an industrial make-and-pack production facility with batch splitting and quality release times. In Z. Lian, Z. Wu, M. Xie, and R. Jiao, editors, *Proc. IEEE Intl. Conf. on Industrial Engineering and Engineering Management*, pages 1230–1234, Macao, China, 2010.
3. P.M. Castro and A.Q. Novais. Scheduling multistage batch plants with sequence-dependent changeovers. *AIChE J.*, 55:2122–2137, 2009.
4. J. Cerdá, G.P. Henning, and I.E. Grossmann. A mixed-integer linear programming model for short-term scheduling of single-stage multiproduct batch plants with parallel lines. *Ind. Eng. Chem. Res.*, 36:1695–1707, 1997.
5. S.J. Honkomp, S. Lombardo, O. Rosen, and J.F. Pekny. The curse of reality - why process scheduling optimization problems are difficult in practice. *Comp. Chem. Eng.*, 24:323–328, 2000.
6. C.A. Méndez, G.P. Henning, and J. Cerdá. An MILP continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities. *Comp. Chem. Eng.*, 25:701–711, 2001.

Scheduling of large dynamic job shop problems by concurrent non-blocking threads

Todor Dimitrov • Michael Baumann • Manfred Schenk

1 Introduction

Job shop scheduling problems are known for their complexity and NP-hardness. A variety of algorithms such as simulated annealing, genetic algorithms, ant colony optimization or neural networks is proposed for finding better and faster solutions for practical and theoretical problems. As these algorithms require a high computational time for producing a satisfying solution, many run-time optimizations have been investigated. One possibility is to incrementally compute data structures such as constraint graphs. Another possibility is to distribute the solving on multiple CPUs or computers in order to speed up the solution finding. Since the job shop problem is known to be hardly decomposable, the second one is more complex. Nowadays every computer and even mobile phones on the market have multi-core CPUs; so the demand of effective distributed optimization algorithms grows.

This work presents an approach to flexible distributing of the job shop scheduling across many independent threads. The individual solver threads work on the same solution but optimize only disjunctive parts of it in parallel. The process of discovery and isolation of the solution parts uses an optimized data structure that holds both the constraint graph and intermediate evaluation objects. The intermediate objects not only form the schedule evaluation but are also used for locating and rating problems in the solution. Multiple threads operate on and incrementally compute the optimized data structure without blocking each other. Human interaction, production messages or other changes are also possible without blocking the operating threads. The aforementioned features make the approach suitable for practical application, too.

2 Related Works

Parallel algorithms have been a subject of research for many years now. A survey of parallel algorithms for global optimization problems was done by Pardalos et al.[1]. This survey covers stochastic algorithms as well as deterministic ones. Another subject of research is the decomposition of the original problem into smaller sub-problems which is needed as a prerequisite for many parallel algorithms. The principles of the decomposition of job shop scheduling problems have been generally addressed in the work of Beck and Gomes [2].

Todor Dimitrov, Michael Baumann, Manfred Schenk

Fraunhofer Institute of Optronics, System Technologies and Image Exploitation, Karlsruhe,
Germany

E-mail: {todor.dimitrov| michael.baumann| manfred.schenk}@iosb.fraunhofer.de

In practical applications it is better to have a near optimal solution after a short period of time than a perfect solution far in the future. Both Krishna et al. [3] and Steinhöfel et al. [4] are trying to achieve this by using a distributed version of simulated annealing. Another goal in practical applications is to avoid turbulence caused by fast changing plans since such plans lose people's acceptance as stated by Wiendahl [5].

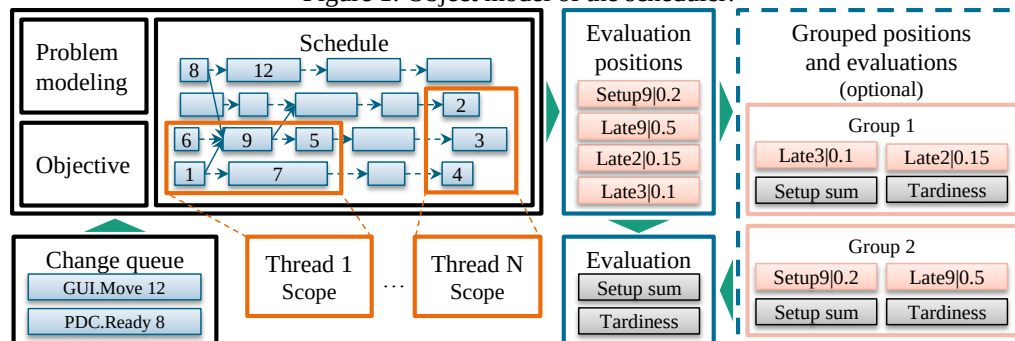
Considering all these constraints, the given problem can be specified as a large dynamic scheduling problem which is processed by iterative algorithms changing only a relatively small part of a schedule per iteration. This is where incremental evaluation takes place: it avoids expensive reevaluation of the whole schedule by calculating the impacts of schedule changes on the evaluation results. Early research on incremental evaluation and the underlying theories has been conducted by Sundaresh and Hudak [6] twenty years ago. About ten years later Liu [7] describes a system for incrementalization. This approach has applications in several areas e.g. interactive systems, optimizing compilers and many others. Incremental evaluation is also used in recent scheduling frameworks like iSchedule [8].

In this work some approaches from Fox et al. [9] are also used, especially their research on conflict handling as part of the OPIS architecture. OPIS uses constraint propagation to categorize problems previously gathered in a kind of pool. These categorized problems are then collected on a "blackboard" and are given some priority value. According to its priority one problem is being selected and repaired by applying some operator chosen by heuristic.

3 Explanation of the approach

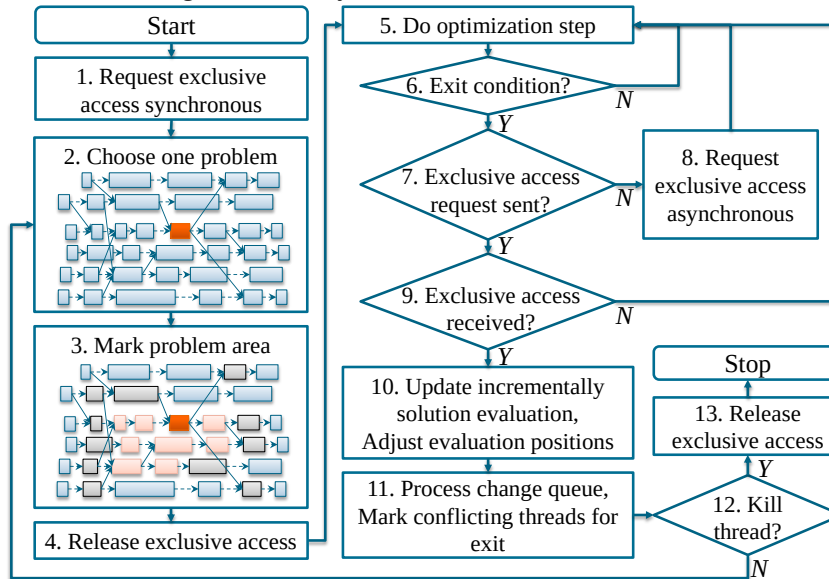
The main idea of this approach is to discover and isolate sub-problems in the whole job shop problem and assign them to concurrent solver threads. Sub-problems may overlap each other, but only disjunctive sub-problems can be solved in parallel. The discovery task works on an optimized data structure that is computed incrementally. An overview of the solution model is shown in figure 1. An incrementally computed constraint graph represents all dependencies between all objects of the job shop problem such as machines, tasks, jobs and tools.

Figure 1: Object model of the scheduler.



The analysis of the solution and the constraint graph produces intermediate evaluation objects (evaluation positions) that are used to describe and measure problems in the solution. The idea is taken from the OPIS AI Scheduler [9]. Such an evaluation position is for example the tardiness of one job. The evaluation positions are then used to compute the evaluation values of the schedule. For example all evaluation positions for tardy orders are being accumulated and the result is the total tardiness of the schedule. As there are a high number of evaluation positions, grouping them according to different criteria like same machine class or same week can speed up the discovery and isolation task. Figure 1 contains an example of the isolation and assignment of two disjunctive parts of the solution to different threads. User interactions (updates to the solution) and production progress messages (updates to the problem model) are stored in the change queue and marked for highest priority processing.

Figure 2: Activity flow chart of one solver thread.



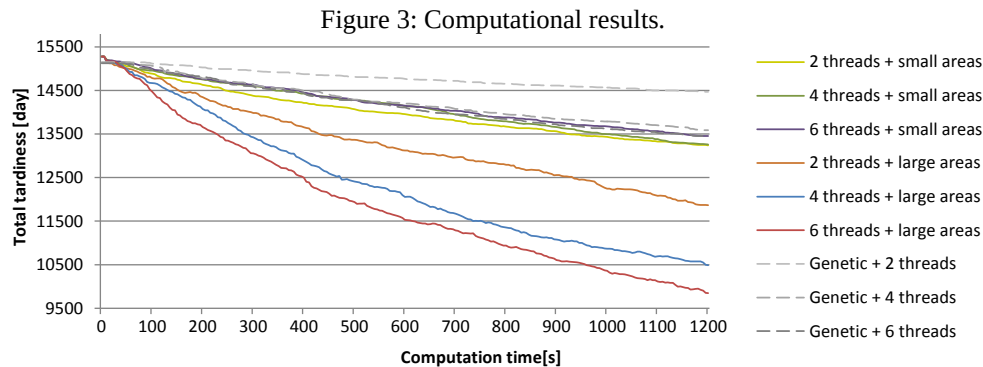
The presented object model can be maintained by multiple independent threads without blocking each other. The lifecycle of one solver thread is shown in figure 2. Firstly, the thread requests and receives exclusive access to the solution (step 1). The exclusive access allows the thread to walk through unmarked areas of the constraint and evaluation positions graph to find a problem (step 2). After finding a problem the thread can estimate and mark the problem area and set the border of the marked area as read-only (step 3). After this the thread can release the exclusive access (step 4). The thread then performs an optimization step (step 5). An optimization step may be one permutation in simulated annealing or a new generation in a genetic algorithm e.g. After each step the thread proves the following exit conditions (step 6): a significant better solution is found; computational time limit is reached; marked items of the problem area prevent the processing of the change queue; the computer needs free resources. If at least one exit condition is met, the thread sends an asynchronous request for exclusive access (step 8) and continues to optimize (step 5) instead of waiting in idle state. After exiting the optimization cycle the thread updates the constraint and evaluation positions graph and updates the schedule evaluation (step 10). If the change queue is not empty, the items of the change queue are processed and those threads, which are preventing items from being updated, are requested to exit their cycle (step 11). If a thread is not requested to terminate, it proceeds with step 2 and chooses another problem area for solving.

As this lifecycle allows the highest utilization of the available computation time, the achieved result highly depends on the size of the marked problem area and on the computation time limit.

4 Computational results

A wafer production factory data was used for evaluating the presented approach. The chosen test case consists of 2200 orders with 13.600 processes and 90 machines. Almost all processes have over 60 alternative machines. The optimization target is minimizing tardiness. Different number of parallel threads was simulated in combination with marked area sizes of 3% and 12% from the whole schedule. Figure 3 shows the mean achieved tardiness of the 10 performed runs per combination for the elapsed computation time. As expected, larger marked problem areas and higher thread count led faster to better results. In comparison with a tuned genetic algorithm the approach achieved better results, even when the evaluation of new generation was distributed on more CPUs. The tests showed that adding more threads to the

approach does not necessarily reduce the performance per single thread. The size of the marked problem areas has a significant impact on the overall approach performance and thus a case-specific fine-tuning is needed.



5 Conclusion

With this approach shorter computing times are possible for large problems. Corresponding to the number of solver threads, better results are reachable in drastically shortened time while the solution space is much better investigated. In addition to the common main storage, this approach has other useful features for dynamic job shop scheduling problems in production systems. Events like new orders, started or stopped processes, tardy processes or machine breakdowns will be immediately integrated in the main storage from one of the participating solver threads without a restart of the optimization process. Solver threads changes and user interactions in multi-user environments will be directly visible for all users. In addition, closer collaboration between manual and automated optimization procedures is achievable.

References

1. Pardalos, P.M. et al., Parallel algorithms for global optimization, in: Solving Irregular Problems in Parallel: State of the Art, Springer-Verlag, Berlin (1995)
2. Beck, H. and Gomes, CP, Principles underlying the decomposition of job-shop scheduling problems, in: First International Conference on Intelligent Systems Engineering (Conf. Publ. No. 360), p. 210—217, (1992)
3. Krishna, K. et al., Distributed simulated annealing algorithms for job shop scheduling, IEEE Transactions on Systems, Man and Cybernetics Vol.25, p. 1102—1109, IEEE, (2002)
4. Steinhöfel, K. et al., Fast parallel heuristics for the job shop scheduling problem, Computers & Operations Research Vol. 29., p. 151-169, Elsevier, (2002)
5. Wiendahl, H.H., Fast and Reliable Order Management Design Using a Qualitative Approach, in: Advances in Production Management Systems, p. 237-244, Springer, (2007)
6. Sundaresh, R.S. and Hudak, P., Incremental computation via partial evaluation, in: Conference Record of the 18th Annual ACM Symposium on POPL, p.1-13, (1991)
7. Liu, L. et al., Differential evaluation of continual queries, in: icdcs, Published by the IEEE Computer Society (1996)
8. Liret, A. et al., iSchedule, an Optimisation Toolkit for Complex Scheduling, MISTA (2007)
9. Smith, S.F. and Fox, M.S. and Ow, P.S., Constructing and maintaining detailed production plans: Investigations into the development of knowledge-based factory scheduling systems, in AI Magazine Vol. 7, p. 45-61, (1986)

Job Scheduling to Minimize Weighted Completion Time Variance on Identical Parallel Machines

Yang Sun • Xueping Li • John W. Fowler

1 Introduction

Completion Time Variance (CTV) is an important performance measure for job scheduling problems in computer and network systems, just-in-time (JIT) production systems, and other application domains where it is important to ensure service stability and predictability. Since the original introduction by Merten and Muller [1], the problem of minimizing CTV on a single machine has received extensive attention. In scheduling language, the problem can be denoted as $1|CTV$ using the triplet notation. Kubiak [2] accomplishes the proof that $1|CTV$ is NP-complete. Eilon and Chowdhury [3] establish a very important property for $1|CTV$ that the optimal solution is in V-shape in terms of job processing time. V-shape means that the smallest job is scheduled in the “middle” of the sequence. Jobs before the smallest job are scheduled in Longest Processing Time first (LPT) order; after the smallest one, in Shortest Processing Time first (SPT) order. Numerous heuristics are proposed in order to find near-optimal solutions for $1|CTV$; many are V-shape-based. See Li et al. [4] for a more detailed review on $1|CTV$ properties and heuristics.

Cai and Cheng [5] study the CTV problem in an identical parallel machine environment and give eight important optimality properties for $Pm|CTV$. The proof for NP-completeness of the $Pm|CTV$ problem is also accomplished [5]. Li et al. [4] propose an efficient heuristic based on the optimality properties of the $Pm|CTV$ problem.

Users’ perceived importance on service stability and predictability may be different. Cai [6] studies the weighted version of the $1|CTV$ problem. Consider a set of $J = \{1, 2, \dots, n\}$ independent jobs with processing times $P = \{p_1, p_2, \dots, p_n\}$ and weights $W = \{w_1, w_2, \dots, w_n\}$. It is assumed that all jobs are available to be processed at time zero, and preemption is not allowed. Let $\pi \in \Pi$ be a feasible schedule, $C_j(\pi)$ be the completion time of job j under π , $\forall j \in J$, and $\bar{C}(\pi)$ be the grand mean of completion times under π . The objective is to find the minimum weighted completion time variance (WCTV):

$$\min_{\pi \in \Pi} \frac{1}{n-1} \sum_{j=1}^n w_j [C_j(\pi) - \bar{C}(\pi)]^2$$

The problem is similar to the classical scheduling problem of minimizing total weighted earliness and tardiness (with a loose common due date). Our problem penalizes the deviation from the mean (or the common due date) with a quadratic function so that a larger deviation

Yang Sun
College of Business Administration, California State University, Sacramento
E-mail: suny@csus.edu

Xueping Li
Department of Industrial & Information Engineering, The University of Tennessee
E-mail: xueping.li@utk.edu

John W. Fowler
School of Computing, Informatics, & Decision Systems Engineering, Arizona State University
E-mail: john.fowler@asu.edu

receives a heavier penalty rate. Cai [6] shows that, if $p_i \geq p_j$ implies $w_i \geq w_j$, the optimal schedules are V-Shaped in terms of weighted processing time. Li et al. [7] show that such a V-Shape property does not hold for 1|WCTV when jobs are arbitrarily weighted. However, a strong V-Shape tendency exists for optimal schedules.

In this research, we address the open problem Pm|WCTV, where jobs are processed on a set of $M = \{1, 2, \dots, m\}$ identical parallel machines. Assume that idle times can be unrestrictedly inserted before or between jobs. The Pm|CTV problem is a special case of Pm|WCTV where all weights are equal. Given that Pm|CTV is NP-complete, the generalized Pm|WCTV is also NP-complete. Such computational intractability prevents people (or computers) from finding the optimal schedule in a reasonable amount of time for large problem instances. Hence, it is necessary to develop heuristics that can provide good schedules quickly. This is important in many application domains. For example, today's data services often have oceans of data packages to process and quick response is a must.

2 Heuristics

Heuristics used in this research for solving Pm|WCTV is listed below. The first three are simple, intuitive heuristics. The forth one is a constructive heuristic based on the strong V-Shape tendency. The fifth and sixth are improving heuristics that start with the solution provided by the forth heuristic. Note that an extra step is taken for all algorithms to synchronize the mean completion time across all machines by inserting idle time before the first job. Let $\bar{C}_i$ be the mean processing time on machine i . Insert idle time $s_{0,1} = \max(\bar{C}_1, \bar{C}_2, \dots, \bar{C}_m) - \bar{C}_i$ before the first job on each machine.

1) First-In-First-Out (FIFO)

FIFO is one of the most commonly used dispatching rules that are used in manufacturing processes as well as computer and network operations. The jobs are put in a random order (or in their original order that is given). Whenever a machine becomes available, the next job in the pool is taken.

2) Weighted Shortest Processing Time first (WSPT)

WSPT is a very good heuristic for identical parallel machine scheduling problems with the objective of minimizing weighted total (or mean) completion time. The jobs are ordered in decreasing order of w_j / p_j and the available machine takes the next job with the largest w_j / p_j among remaining jobs.

3) Weighted Longest Processing Time first (WLPT)

This is a dispatching rule that *approximately* balances the workloads on machines in terms of total weighted job processing time. The jobs are ordered in increasing order of w_j / p_j .

4) Balanced V-Shape (BVS) Heuristic

We first assign jobs to machines according to the WLPT rule to approximately balance the loads on machines. On each machine, jobs are originally sequenced in increasing order of w_j / p_j . Put jobs on odd positions (i.e., jobs 1, 3, 5...) in Set 1 and jobs on even positions (jobs 2, 4, 6...) in Set 2. Sequence Set 1 jobs in WLPT order followed by Set 2 jobs in WSPT order. That is, jobs are re-sequenced into a V-shape in terms of weighted processing time on each machine.

5) Homologous Pair-wise Interchange (HPI) based on BVS

The solution from the BVS heuristic is taken as the starting point, and improvement is obtained via pair-wise interchange of jobs on homologous positions. That is, we keep swapping jobs on equivalent positions across different machines, as well as jobs on antithetical positions on the same machine, until no more improvement can be obtained.

6) Arbitrary Pair-wise Interchange (API) based on BVS

The API heuristic is similar to the HPI heuristic. The difference is that interchanges are made for all possible pairs among all jobs.

Both API and HPI are composite heuristics starting from the BVS solution.

7) Complete Enumeration

Complete enumeration is used to get *optimal* solutions for small problem instances.

3 Conclusion Remarks Based on Computational Tests

Twenty (20) small problem instances with $n = 8$ and $m = 2$ are randomly generated to test the proposed algorithms above. The processing time is uniformly distributed between 1 and 100, and the weights are equally likely to be 1, 2 or 4, which represent normal jobs, important jobs, and extremely important jobs, respectively. Table 1 shows the results for the 20 instances and summarizes the average performances for different algorithms. WCTVs obtained from different heuristic algorithms are divided by the optimal-WCTV, obtained via complete enumeration, for each instance. Therefore, the optimal schedule always has a performance score of 1. Performance scores of the other algorithms are greater than or equal to 1.

Twenty (20) large problem instances with $n = 18$ and $m = 3$ are also generated. The search space is fairly large with this setting and complete enumeration cannot provide an optimal solution in a reasonable amount of time. WCTVs obtained from the other algorithms are divided by the WCTV of the API solution for each of the large instances. See Table 1.

Table 1 Computational Test Results

Test Results for Small Problem Instances								Test Results for Large Problem Instances							
Instance	FIFO	WSPT	WLPT	BVS	HPI	API	Optimal	Instance	FIFO	WSPT	WLPT	BVS	HPI	API	
1	2.147	2.93	1.488	1.495	1.15	1	1	21	2.585	2.657	1.951	1.088	1.018	1	
2	6.265	5.925	1.553	1.108	1.108	1.029	1	22	2.243	2.41	1.689	1.234	1.075	1	
3	2.603	2.603	1.459	1.018	1.004	1.004	1	23	1.624	1.876	1.472	1.033	1.022	1	
4	2.27	2.585	1.263	1.261	1.248	1	1	24	3.601	3.983	2.116	1.142	1.049	1	
5	2.758	1.948	1.273	1.621	1.135	1	1	25	1.709	3.312	1.762	1.122	1.043	1	
6	2.061	2.707	1.28	1.358	1.039	1.039	1	26	3.22	3.053	1.834	1.016	1.011	1	
7	2.764	3.956	2.091	1.149	1.01	1	1	27	3.443	4.619	2.233	1.77	1.036	1	
8	2.861	2.526	1.691	1	1	1	1	28	2.659	2.912	1.914	1.078	1.058	1	
9	4.086	5.842	1.92	1	1	1	1	29	4.096	3.867	1.628	1.139	1.023	1	
10	1.968	3.328	1.163	1.096	1.096	1.039	1	30	3.35	3.358	1.963	1.098	1.036	1	
11	2.354	3.804	1.582	1.208	1.124	1	1	31	2.756	3.191	1.366	1.141	1.06	1	
12	1.481	1.852	1.57	1.286	1.167	1.086	1	32	3.074	3.511	1.708	1.217	1.029	1	
13	1.971	2.024	1.777	1.425	1	1	1	33	3.316	3.135	1.361	1.135	1.048	1	
14	2.844	3.093	1.543	1.322	1	1	1	34	2.5	2.782	1.509	1.166	1.062	1	
15	1.406	2.398	1.48	1.205	1.099	1.023	1	35	2.469	3.621	1.686	1.161	1.028	1	
16	3.084	6.349	3.132	3.588	1.178	1.01	1	36	2.749	3.526	2.211	1.144	1.057	1	
17	1.776	2.918	1.43	1.493	1.287	1.017	1	37	2.357	3.197	1.595	1.155	1.059	1	
18	2.337	2.24	1.386	1.158	1.048	1	1	38	2.997	3.301	2.178	1.133	1.013	1	
19	2.186	3.718	1.687	1.363	1.217	1.014	1	39	3.12	2.757	2.059	1.18	1.039	1	
20	3.645	3.918	1.745	1.312	1.071	1.014	1	40	2.449	3.257	1.692	1.15	1.017	1	
Average	2.643	3.333	1.626	1.373	1.099	1.014	1	Average	2.816	3.216	1.796	1.165	1.039	1	

It appears that API is the best among all proposed heuristics. While API is also the slowest among proposed algorithms, for tested large instances it takes less than 1 second for API to get the solution on a PC with a 1.9MHz Pentium M CPU and 1GB RAM in the MATLAB computing environment – this can be easily improved in a different computing environment. API provides optimal or near-optimal solutions for small problem instances. Although it is difficult to assess how far the performance of API is away from the optimal for large problem instances, it is convincing that API is a very good heuristic for solving $Pm|WCTV$.

Future research is suggested for non-identical parallel machines as well as the restrictive version of the problem in which idle times are not allowed and all machines must start processing jobs at time zero. In addition, both mean and variance measures can be considered at the same time in a multi-criteria scheduling problem.

Reference

1. Merten, A.G. and M.E. Muller, Variance minimization in single machine sequencing problems, *Management Science*, Vol. 18, 518-528 (1972)
2. Kubiak, W., Completion time variance on a single machine is difficult, *Operations Research Letter*, Vol. 12, 49-59 (1993)
3. Eilon, S. and I.G. Ghowdhury, Minimizing waiting time variance in the single machine problem, *Management Science*, Vol. 23, 567-575 (1977)
4. Li, X., Y. Chen and Y. Sun, Minimizing job completion time variance for service stability on identical parallel machines, *Computers and Industrial Engineering*, Vol. 58, 729-738 (2010)
5. Cai, X. and T.C.E. Cheng, Multi-machine scheduling with variance minimization, *Discrete Applied Mathematics*, Vol. 84, 55-70 (1998)
6. Cai, X., Minimization of agreeably weighted variance in single machine systems, *European Journal of Operational Research*, Vol. 85, 576-592 (1995)
7. Li, X., N. Ye, T. Liu and Y. Sun, Job scheduling to minimize the weighted waiting time variance of jobs, *Computers and Industrial Engineering*, Vol. 52, 41-56 (2007)

Automated scheduling at a joinery site

Pieter Smet · Wim Vancroonenburg · Greet
Vanden Berghe

1 Introduction

Small to medium size joineries construct windows on a make to order basis. Commonly, windows are ordered from a fixed set of types that can then be tailor-made, e.g. according to custom size, materials, colour, extra decorative profiles, etc. Typically, smaller joineries are the most flexible: they operate in a *job shop* fashion with multiple unconnected workcentres that perform specific tasks like cutting profiles, welding frames, assembling windows, etc. Medium size joineries have a more organized *flow shop* layout. Their workcentres are grouped into *lines*, which are connected to transport systems and intermediate buffers. Although still very flexible due to many interconnections between lines, the flow is, to a larger extent, determined by the layout. Operations such as cutting profiles and welding frames are carried out at the start, while operations such as final assembly and finishing come at the end.

We focus on scheduling operations in joineries in which workcentres are organized in a flexible flow shop layout. Different types of windows need to be produced, ranging from simple (e.g. windows consisting of four profiles in a rectangular frame) to complex (e.g. windows consisting of an outer frame, and three inner frames of which one can be opened). In this context we identify a scheduling problem that is composed of several jobs, each requiring multiple operations some of which can be performed in parallel. Jobs can differ both in number and type of operations. Finally, job operations can be performed on multiple workcentres and in different *modes*, where each mode has a different duration and requires specific resources (Sprecher et al. [1]).

We present a solution method that combines a serial schedule generation scheme and a local search algorithm for solving real world scheduling problems at joineries. Our approach provides solutions in a very short amount of time compared to the time required by human planners.

Pieter Smet, CODES, KAHO Sint-Lieven
Wim Vancroonenburg, CODES, KAHO Sint-Lieven
Greet Vanden Berghe, CODES, KAHO Sint-Lieven & Dept. of Computer Science, K.U.Leuven
E-mail: {pieter.smet, wim.vancroonenburg, greet.vandenberghel}@kahosl.be

2 Problem description

A production environment layout is given, consisting of capacity constrained workcentres and buffers. These workcentres and buffers can be seen as vertices on a directed graph (Figure 1), where the edges indicate that a part can move from one location to another (e.g. from a workcentre to a buffer, from a workcentre to another workcentre, etc.). Furthermore, a set of orders is defined for the production of possibly different windows. The orders have a release date and a due date. A window is described as an assembly of several parts which can be decomposed further into several subparts, thus forming a hierarchy of parts that need to be manufactured. This hierarchy imposes precedence constraints on the construction of the parts, i.e. a part can only be assembled after its subparts have been constructed. Each individual part requires a number of operations, which can be performed at one or more workcentres. The problem is then to find a feasible schedule, i.e. respecting precedence constraints and workcentre availability, for all the required operations, that minimizes a particular objective (e.g. total makespan, total weighted tardiness or any combination of real world objectives).

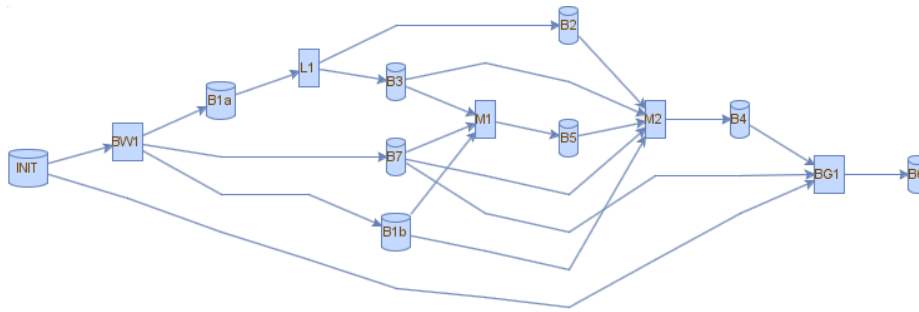


Fig. 1 Example of a flow shop layout.

One important issue is that workcentres are strictly bound by routing constraints. This implies that when the choice is made to perform an operation on a part at a certain workcentre, it constrains the next operation on that part to be performed at a workcentre that is connected to the preceding one. The hierarchical composition of assemblies imposes additional constraints on the selected routes for parts. If a part e.g. has two subparts, then the routes for both subparts should eventually arrive at the same location, where the final part is assembled.

We focus on the offline version of the problem, i.e. at a certain point in time, a set of orders needs to be scheduled in the upcoming period. The online version of the problem, where orders are added continuously, is not explicitly discussed. However, this version of the problem can also be tackled by our approach. To do this, the schedule is first fixed at the time a new order arrives and this new order is then scheduled in the (fixed) existing schedule.

3 Solution method

The scheduler was designed to be used on problems of varying size. Therefore we opted for an approach that ensures computational feasibility even for very large instances. The presented approach consists of several stages, in which we use a serial schedule generation scheme and a steepest descent heuristic.

Given a set of orders, we first determine a route for each assembly of each order. A recursive algorithm starts from a given part and enumerates all feasible paths for that part through the production layout. A path is feasible for a part, if 1) all the required operations of that part can be performed along the path, and 2) all its subparts have feasible paths that *connect* to the path. The second condition is checked by recursively calling this algorithm for all subparts of the considered part. After enumeration of all possible routes, one route is selected in a heuristic way.

After having determined a route for all assemblies for all orders, we know exactly which operations need to be performed where, and which precedence constraints exist between operations due to the layout constraints. We can then formulate the problem of finding a schedule as a multi-mode, multi-project, resource constrained project scheduling problem (MMMPPRCSP) (Wauters et al. [3]). An order corresponds to a *project* and the operations of all assemblies constitute the project *activities*. The work-centres where the operations are executed, are mapped onto the *resources* required by the activities.

We address this scheduling problem using priority lists and a serial schedule generation scheme (SSGS), which has been shown to be a successful approach for project scheduling problems (Kolisch [2]). First, a priority list is constructed containing the operations (and the mode in which they will be executed) that need to be scheduled. This list is constructed in a heuristic way and is *precedence feasible*, i.e. operations appear in the list later than their predecessors and before their successors. Using a SSGS, this list can then be decoded into a feasible schedule.

In order to improve the quality of the generated schedule, we search through the space of priority lists using a stochastic steepest decent heuristic. Starting from the initial priority list, a list is constructed containing feasible moves to neighbouring solutions. Two types of moves are considered: *shift*-moves, where one operation is moved up or down the priority list, and *change mode*-moves, in which the mode of the operation is changed. For each type of move, a *tournament factor* is defined determining the number of neighbouring solutions that will be explored. We limit the number of explored neighbouring solutions since each evaluation requires the priority list to be decoded using the SSGS, which becomes computationally costly for larger problem instances. From the set of explored solutions, the best improving solution is selected as the starting point for the next iteration of the local search procedure. If no improving solution is found, the procedure continues to the next iteration, keeping the current solution. This is repeated until a stop criterion is met.

The performance of the automated scheduler is dependent on a number of parameter settings. The objective function can combine tardiness, total makespan or makespan per project. When priorities of projects are taken into account, a weighted function can be constructed defining how much tardiness is penalized for each project. The particular objectives can be influential on the effectiveness of the scheduler. Although the steepest descent algorithm is fairly straightforward, a second set of parameters that affect the algorithms' performance can be identified. These parameters include

the tournament factor, the stop criterion (e.g. maximum number of iterations or run time) and the neighborhoods explored by the algorithm.

4 Experiments

A number of experiments have been set up to evaluate the performance of the automated scheduler. These experiments were conducted on instances derived from real world data provided by an industry partner.

Two quantitative measures are used to illustrate the effectiveness of the scheduler: 1) the makespan (total or per project), and 2) the number of projects finished before their due date. In addition, they can be used to measure the quality of solutions obtained with different objective functions. The objectives resulting in schedules that best match the company's objectives can then be identified.

In order to determine the best scheduling horizon, another experiment is set up. Each day of the scheduling period, a schedule is generated from week one to week two and from week three to week four. In addition, the schedule from week one to week four is also generated each day. Based on the analysis of the quantitative measures, the scheduling horizon that induces the best algorithmic performance can be determined.

5 Decision support for operational management

Apart from optimizing the daily operations at a joinery, the scheduler enables the user to perform *what-if analyses*. This practice can be used to simulate what would have been e.g. the extra profit if a certain action would have been undertaken. Possible scenarios for what-if analyses include adding or removing employees, training an employee, adding an extra machine (which may have improved capabilities) or replacing a group of machines with one new machine. For operational or financial managers, it could be interesting to check whether the additional profit after investments can cover the incurred expenses. Another interesting application of the scheduler allows to identify the bottlenecks both in the computer generated and in the executed schedules. When the bottlenecks are different, inefficient use of the production lines is indicated.

The results of the full experimental setup and illustrations of alternative applications, will be presented at the conference. Initial experiments have shown that the automated scheduler only requires a very short amount of time to generate a feasible schedule, in contrast to approximately two days per week required by a manual planner. This improvement already illustrates the added value of the scheduler in practice.

Acknowledgements This research was carried out within the IWT project (IWT 090294).

References

1. A. Sprecher, S. Hartmann and A. Drexler, An exact algorithm for project scheduling with multiple modes, *OR Spectrum*, 19(3), 195-203 (1996)
2. R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, *European Journal of Operational Research*, 90, 320-333 (1996)
3. T. Wauters, K. Verbeeck, P. De Causmaecker and G. Vanden Berghe, A game theoretic approach to decentralized multi-project scheduling, *Proc. of AAMAS10*, 1415-1416 (2010)

Optimal Job Insertion in the No-Wait Job Shop

Reinhard Bürgy · Heinz Gröflin

1 Introduction

The No-Wait Job Shop (NWJS) considered here is a version of the job shop scheduling problem where after completion of an operation, the job's next operation must be started immediately. We study a slight generalization of this problem (still called NWJS) where for any two operations of a job a fixed time lag between their starting times is given. Additionally, we allow for sequence-dependent set-up times between consecutive operations on a machine. The NWJS problem consists in finding a feasible schedule of all operations that minimizes the makespan.

Figure 1 displays in a Gantt chart an example with five jobs and seven machines. The number $j.r$ in a strip refers to the r -th operation of job j . Narrow strips correspond to set-up times.

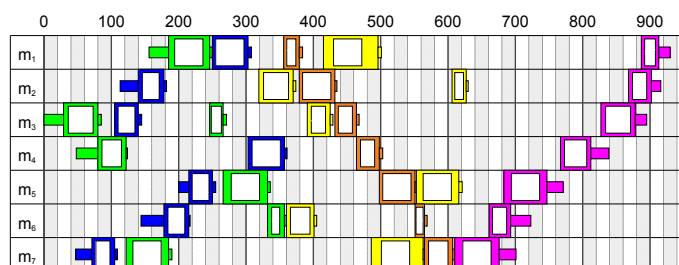


Fig. 1 Example with 5 jobs and 7 machines.

Reinhard Bürgy
Dept of Informatics, University of Fribourg, Bd de Pérolles 90, 1700 Fribourg, Switzerland
E-mail: reinhard.buergy@unifr.ch

Heinz Gröflin
Dept of Informatics, University of Fribourg, Bd de Pérolles 90, 1700 Fribourg, Switzerland
E-mail: heinz.groeflin@unifr.ch

While the Optimal Job Insertion problem is NP-hard in the classical Job Shop, it was shown by Gröflin & Klinkert [3] that it is solvable in polynomial time in the NWJS. We present here an efficient algorithm for this problem. The algorithm is based on a compact formulation of the NWJS and a characterization of all feasible insertions as stable sets (of prescribed cardinality) in a comparability graph. We sketch an application of our optimal job insertion algorithm by devising a competitive heuristic for the NWJS problem.

2 A Compact Disjunctive Graph Formulation

We derive a compact disjunctive graph formulation of the NWJS problem. A sketch of this formulation is presented here. Let $\mathcal{J}$ be the set of all jobs. A *Disjunctive Job-Graph* $F = (\mathcal{J}^+, B, U, c)$ is built as follows. Node set $\mathcal{J}^+$ consists of a node for each job $J \in \mathcal{J}$ and a fictive start node σ and end node τ . We identify these nodes with J, σ and τ . The conjunctive arc set B comprises the arcs (σ, J) and (J, τ) for each $J \in \mathcal{J}$. The set U of disjunctive arcs is defined as follows. Between any distinct nodes $J, K \in \mathcal{J}$, q_{JK} pairs of disjunctive arcs $(J, K)_p, (K, J)_p, p = 1, \dots, q_{JK}$ are introduced. The pair $[J, K]_p := \{(J, K)_p, (K, J)_p\}$ represents a “feasible way” of positioning job J and K with respect to each other. The number q_{JK} of these positions is “small”. The length c_{JK}^p of arc $(J, K)_p$ reflects the “distance” from the first operation of job J to the first operation of job K in position p . Let P be the set of all disjunctive arc pairs, i.e. $P = \{[J, K]_p : J, K \in \mathcal{J}, J \neq K \text{ and } 1 \leq p \leq q_{JK}\}$, and for any distinct J and $K \in \mathcal{J}$, let $D_{JK} = \{[J, K]_p : 1 \leq p \leq q_{JK}\}$. An F -selection is any set $T \subseteq P$ of arc pairs and $U_T \subseteq U$ denotes the set of all arcs used by T . An F -selection T is *complete* if $T \cap D_{JK} \neq \emptyset$ for all distinct $J, K \in \mathcal{J}$. T is *positive acyclic* if $F(T) = (\mathcal{J}^+, B \cup U_T, c)$ contains no cycle of positive length, and *feasible* if it is complete and positive acyclic.

We show that the NWJS can be formulated as the following problem in the Disjunctive Job-Graph F : “Among all feasible F -selections, find an F -selection T minimizing the length of a longest path from σ to τ in subgraph $F(T)$.”

Figure 2 depicts the disjunctive job-graph F for the example of Figure 1.

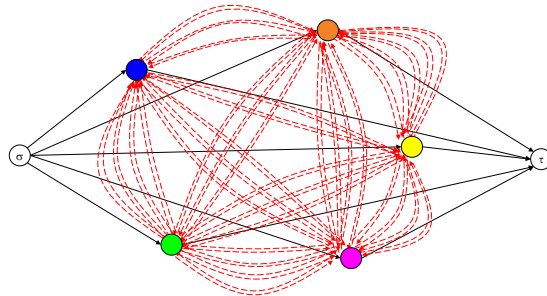


Fig. 2 Disjunctive Job-Graph F of the Example.

3 Optimal Job Insertion in the NWJS

Inserting optimally a job J can be thought of as the following problem. Given a feasible schedule for all other jobs, insert job J in such a way that the resulting schedule is feasible

and its makespan is minimal. In the framework of the compact disjunctive graph formulation a given schedule of all other jobs is specified by an F -selection R that is feasible on $U - \delta(J)$. The disjunctive graph $F^J = (\mathcal{J}^+, B \cup U_R, U^J, c)$, where $U^J = U \cap \delta(J)$, is the *insertion graph* for job J and an F^J -selection $T \subseteq U^J$ is an *insertion* of job J .

The *Optimal Job Insertion Problem* can be stated as follows: “Among all feasible insertions, find an insertion T minimizing the length of a longest path from σ to τ in subgraph $(\mathcal{J}^+, B \cup U_R \cup U_T, c)$.”

3.1 The Conflict Graph H^ρ

We introduce conflict graphs which allow to characterize feasible insertions. Let ω be the makespan of the schedule of all other jobs. For any $\rho > \omega$, the conflict graph at ρ is the graph $H^\rho = (W^\rho, Y^\rho)$ with node set W^ρ and arc set Y^ρ roughly defined as follows. W^ρ contains a node for all “positions” $p = 1, \dots, q_{JK}$ and $K \in \mathcal{J} - J$ which do not imply a makespan $\geq \rho$. There is an arc between two nodes if the corresponding “positions” either imply a cycle of positive length in the insertion graph or a makespan $\geq \rho$.

Let $\mathcal{F}^\rho$ be the family of all feasible insertions T of makespan $\omega(T) < \rho$. We show that there is a one-to-one correspondence between feasible insertions $T \in \mathcal{F}^\rho$ and the stable sets of cardinality $|\mathcal{J}| - 1$ in conflict graph H^ρ .

Then we show that H^ρ is a so-called comparability graph. Finding a maximum size stable set in a comparability graph is achieved in polynomial time for instance via network flows. Thus the following general approach for solving the Optimal Job Insertion Problem is suggested. In an initialization step, let T be the insertion placing J after all other jobs. Set $\rho := \omega(T)$. Then execute the following loop. Determine a maximum size stable set T' in H^ρ . If the stability number $\alpha(H^\rho) := |T'| < |\mathcal{J}| - 1$ then *stop*: T is an optimal insertion. Else reset $T := T'$ and $\rho := \omega(T)$.

We propose a substantially more efficient implementation which exploits further properties of stable sets in the conflict graphs, in particular that the maximum size stable sets of a comparability graph form a lattice.

Without going into further details, we mention here the complexity $\mathcal{O}(n \cdot \max\{n^2, \sum_{K \in \mathcal{J} - J} q_{JK}\})$ of our Optimal Job Insertion algorithm, denoting by n the number of jobs. We remark that in the “classical” case, where a job is assumed to have at most one operation on a given machine, this complexity can be related to the number m of machines. Then $\sum_{K \in \mathcal{J} - J} q_{JK} \leq (n - 1)(m + 1)$, and the Optimal Job Insertion Algorithm runs in time $\mathcal{O}(n^2 \cdot \max\{n, m\})$.

4 An Application and Some Numerical Results

Based on the optimal job insertion algorithm, we developed a heuristic for the NWJS.

A feasible initial selection is built by sequentially inserting each job using the optimal job insertion algorithm. Then, local search is used to improve the initial selection. A neighbor selection is generated by extracting and (re-) inserting optimally a job. This procedure is repeated with various initial selections (generated by randomly chosen job insertion orders) until a computation time limit $maxT$ is reached.

The sketched algorithm has been implemented in Java and run on a PC with 2.66 GHz processor and 4 GB memory. It has been extensively tested on well-known benchmark in-

stances for job shop scheduling problems which are interpreted as no-wait instances. For each instance five runs were performed.

Table 1 displays our numerical results of the instances la26-la35 without set-up times, together with benchmarks of Bożejko and Makuchowski [1] (*BM*), van den Broek [7] (*vdB*) and Zhu et al. [8] (*Zhu*). *BM*, *vdB* and *Zhu* represent, to our knowledge, the best current benchmarks. We tried to use the same computation time as reported by these authors. For this reason, Table 1 has been divided into two blocks. The first block (column 2-4) reports our average results *avg1* and results of *BM* with *their* run-time *T1* seconds. The second block of four columns compares our average results *avg2*, results of *vdB* and results of *Zhu* with their run-time *T2* seconds. On average, our solutions are 2.5%, 4.5% and 5.9% better than those of *BM*, *vdB* and *Zhu*, respectively.

instance	avg1	BM	T1	avg2	vdB	Zhu	T2
(20 x 10)							
la26	2626	2738	14	2526	2598	2707	812
la27	2790	2794	27	2699	2755	2838	834
la28	2697	2741	24	2586	2722	2752	810
la29	2495	2596	12	2418	2427	2539	778
la30	2662	2791	12	2582	2572	2743	822
(30 x 10)							
la31	3718	3869	151	3629	3708	3884	2588
la32	4081	4045	176	4016	4337	4259	2698
la33	3654	3751	120	3594	3976	3842	2587
la34	3814	3936	102	3713	4161	3932	2754
la35	3838	3918	120	3729	3945	3984	2615

Table 1 Comparison of our numerical results *avg1*, *avg2* with benchmarks *BM*, *vdB* and *Zhu* (bold: best)

In summary, we characterized the family of all feasible job insertions in the NWJS and proposed an efficient algorithm for optimal job insertion. Besides structural and computational complexity results, our development proved also valuable in devising a competitive heuristic for the NWJS.

References

1. W. Bożejko and M. Makuchowski. A fast tabu search algorithm for the no-wait job shop problem. *Computers & Industrial Engineering*, 56:1502–1509, 2009.
2. J. Framinan and C. Schuster. An enhanced timetabling procedure for the no-wait job shop problem: a complete local search approach. *Computers & Operations Research*, 33(5):1200–1213, 2006.
3. H. Gröflin and A. Klinkert. Feasible Insertions in Job Shop Scheduling, Short Cycles and Stable Sets. *European Journal of Operations Research*, 177:763–785, 2007.
4. N. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525, 1996.
5. A. Mascis and D. Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.
6. C. Schuster. No-wait job shop scheduling: Tabu search and complexity of subproblems. *Mathematical Methods of Operations Research*, 63(3):473–491, 2006.
7. J. van den Broek. *MIP-based Approaches for Complex Planning Problems*. PhD thesis, Technische Universiteit Eindhoven, 2009.
8. J. Zhu, X. Li, and Q. Wang. Complete local search with limited memory algorithm for no-wait job shops to minimize makespan. *European Journal of Operational Research*, 198(2):378–386, 2009.

A GRASP Approach for Scheduling Jobs with Incompatible Families on Parallel Batch Machines

Christian Almeder •Lars Mönch

1 Introduction and Problem

This research focuses on the scheduling of batch processing machines found in the diffusion and oxidation areas of a semiconductor wafer fabrication facility (wafer fab). The processing times of these batching operations are extremely long (10 hours) compared to other operations (1-2 hours) in a wafer fab. Though several jobs can be processed simultaneously on these batch processing machines, process restrictions require that only jobs belonging to the same family can be processed together at a time. We have $f = 1, \dots, F$ incompatible families. The family of job j is denoted by $s(j)$. Jobs of the same family have the same processing time on the batch processing machines. The maximum number of jobs that can be processed within one batch is called the maximum batch size. We denote the maximum batch size of machine i by B_i , i.e., the maximum batch size can differ from machine to machine. Machine dedications are assumed, i.e., only batches of specific families can be processed on a given machine. Usually each family can be processed on several different machines. In addition, the jobs to be processed have different ready times in order to reflect differences in arrival times of the jobs. In the presence of unequal ready times, it is sometimes advantageous to form a non full batch while in other situations it is a better strategy to wait for future job arrivals in order to increase machine utilization.

We model diffusion and oxidation operations as parallel batch processing machines with incompatible job families. The performance measure to be minimized is total weighted tardiness (TWT). TWT is the summation of the weighted tardiness $w_j T_j$ over all jobs $j = 1, \dots, n$, where $T_j = \max(C_j - d_j, 0)$. Here, w_j is the weight of job j , C_j is the completion time, and d_j the due date of job j . Using the $\alpha | \beta | \gamma$ notation, the researched problem can be represented as follows:

$$R | p\text{-batch, incompatible}, M_j, B_i | TWT, \quad (1)$$

where we denote by R unrelated parallel machines and $p\text{-batch, incompatible}$ refers to the batch processing with incompatible families. The notation M_j is used to indicate that job j can be processed only on any machine belonging to the specific subset M_j of the set of all parallel machines. The notation B_i is utilized to refer to the machine specific maximum batch sizes. Problem (1) is NP-hard because the single machine problem $1 || TWT$ which is known to be NP-hard can be reduced to problem (1). Therefore, we have to look for efficient heuristics to solve large-scaled problem instances within reasonable amount of time. Note that problem (1) is a generalization of the problem researched in [4], where genetic algorithms are proposed. A

Christian Almeder
Vienna University of Economics and Business
E-mail: christian.almeder@wu.ac.at

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

memetic algorithm is proposed in [1] for the problem studied in [4]. A Greedy Randomized Adaptive Search Procedure (GRASP) approach is presented for $P|p\text{-batch}|C_{max}$ in [2]. A MIP formulation is applied in [3] to solve the subproblems that result from the time window decomposition approach for problem (1). Furthermore, a Variable Neighborhood Search (VNS) approach is studied in the same paper. In this paper, we propose a GRASP approach. It is a multi-start procedure in which each iteration consists of a construction phase and a local search phase [5]. To the best of our knowledge, this metaheuristic has not been used so far to tackle problem (1).

2 Time Window Decomposition Technique

We summarize a list scheduling approach for problem (1) based on time window decomposition (TWD) scheme because the construction phase of our GRASP approach is based on this heuristic. The following four steps are performed (cf. [3] for details).

1. When a machine becomes available at time t , only jobs with ready times smaller than $t + \Delta t$ are considered. We use the set $\tilde{J}_f(t) := \{j \mid r_j \leq t + \Delta t, s(j) = f\}$ for each family instead of the set of all jobs. We sequence all jobs within $\tilde{J}_f(t)$ with respect to the ATC dispatching rule in non-increasing order. Then, we consider only the first *thresh* jobs from the sorted list to form batches. The resulting set is called $\hat{J}_f(t)$.
2. Consider all batch combinations from jobs of $\hat{J}_f(t)$. Each potential batch is assessed using the BATC-II batching rule.
3. The batch with the largest BATC value is chosen among the different families.
4. The time $t' \leq t + \Delta t$ of the next event is calculated, i.e., a machine becomes available or a new job is arrived. Go to Step 1 when there are still jobs not sequenced, otherwise Stop.

3 Local Search Approach

We propose three different neighborhood structures used in the local search approach. Two of them are job-based, i.e., jobs are relocated to different batches in the final schedule. The third neighborhood structure operates purely on the batch level trying to relocate whole batches.

Job insert: A job is moved to another non full batch of the same family or a new batch is formed at the end of the schedule. Only batches which start after the ready time of the job are considered.

Job swap: Two jobs of the same family in different batches are exchanged.

Batch swap: Two batches of the same family on different machines are exchanged. Both batches must be within the limits given by the maximum batch sizes B_i of the two machines.

In Section 5, we report on test results on each neighborhood structure alone as well as on a combination where all three neighborhood structures are applied iteratively, i.e., each neighborhood is used exhaustively in the local search procedure as long as the solution improves. If no improvement is possible we switch to the next neighborhood.

4 GRASP Approach

The GRASP approach is an iterative procedure. A construction phase and a local search phase are used within each step of a single iteration. We consider two strategies to randomize the construction heuristic that correspond to Step 1 and Step 3 of TWD. The first strategy is based on the idea that the different jobs sets $\hat{J}_f(t)$ might lead to different batches. Therefore, we ex-

change c jobs randomized from $\tilde{J}_f(t) \setminus \hat{J}_f(t)$ and swap them with jobs from $\hat{J}_f(t)$, i.e., the outcome of Step 1 of TWD is randomly. The second strategy is related to Step 3. Here, we select randomly a batch among the batches with largest BATC value of the different families according to a discrete probability distribution with probability weights that are given as the ratio of the corresponding BATC index of the batch and the sum of the BATC indices of the selected batches of the remaining families at time t . The first strategy outperforms the second one. Therefore, we use the first one within the computational experiments. Furthermore, the combined local search procedure using all three neighborhoods described in Section 3 is utilized.

5 Computational Results

Our computational results are based on the design of experiments used in [3]. It includes a different number of machines, jobs per family, number of families, wide and tight due dates, and different dedication settings. Totally, we consider 162 problem instances. We tested all three neighborhoods, the combined version, and the GRASP. The results of computational experiments are shown in Table 1. We set $\Delta t = \bar{p}/4$, $thresh=15$, and $c=4$ based on some preliminary experimentation. Here, $\bar{p}$ is the average processing time of all jobs. The number of GRASP iterations is 1000. We also use a MIP formulation to check the correctness of the implementation for small-sized problem instances.

Table 1: Computational Results for Problem (1)

Compare	Number of machines			Jobs per family			Number of families			Total
	m = 3	m = 4	m = 5	n = 20	n = 30	n = 40	F = 4	F = 6	F = 8	
BATC-II	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Job insert	0.990	0.995	0.997	0.995	0.993	0.995	0.997	0.993	0.992	0.994
Job swap	0.987	0.988	0.988	0.988	0.988	0.986	0.985	0.988	0.990	0.988
Batch swap	0.998	0.999	0.997	0.998	0.998	0.998	0.998	0.998	0.997	0.998
Combined	0.976	0.983	0.983	0.983	0.980	0.980	0.981	0.981	0.980	0.981
GRASP	0.897	0.953	0.939	0.929	0.923	0.912	0.916	0.895	0.941	0.926

We can see from Table 1 that the combined version of the local search is advantageous. The solution quality gained is almost the sum of the improvements gained by each neighborhood individually. Furthermore, we note that GRASP clearly outperforms the schedules obtained by BATC-II. The GRASP approach provides a similar solution quality as the VNS approach proposed in [3]. However, the computing time for VNS was one minute per instance on an Intel Pentium 4 CPU 3.40GHz, 2 GB RAM computer, while GRASP needs on average slightly more time per instance.

References

1. Chiang, T.-C., Cheng, H.-C., Fu, L.-C., A memetic algorithm for minimizing total weighted tardiness on parallel batch machines with incompatible job families and dynamic job arrivals. *Computers & Operations Research*, 37, 2257-2269, (2010).
2. Damodaran, P., Velez-Gallego, M. C., Maya, J., A GRASP approach for makespan minimization on parallel batch processing machines. *Journal of Intelligent Manufacturing*, accepted for publication (2009)
3. Klemmt, A., Weigert, G., Almeder, C., Mönch, L. A comparison of MIP-based decomposition techniques and VNS approaches for batch scheduling problems. *Proceedings of the 2009 Winter Simulation Conference*, 1686-1694, (2009)

4. Mönch L., Balasubramanian, H., Fowler, J., Pfund, M., Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. *Computers & Operations Research*, 32, 2731-2750 (2005)
5. Resende, M. G. C., Ribeiro, C. C., Greedy randomized adaptive search procedures. In: *Handbook of Metaheuristics*, Springer, Berlin, 219-249, (2003)

Minimizing Total Weighted Tardiness and Total Processing Costs for the Scheduling with Alternative Process Plans

R. Čapek · P. Šůcha · Z. Hanzálek

1 Introduction

This research is motivated by the production processes that involve several ways how to complete a product (e.g. wire harnesses production in Styl Plzeň). In other words, more process plans can be defined for one type of a product. Since only one of these process plans has to be chosen, we call them *alternative process plans*. Traditional scheduling approach (see [1]) assumes an exactly given set of activities and therefore there is only one process plan. We use an approach extended by a formulation of alternative process plans where so called *alternative branchings* can be defined to model the possibility of choice. In our approach, both the traditional scheduling and the decision which activities will be present in the schedule are integrated into one problem.

In this paper, we focus on the scheduling on dedicated machines considering sequence dependent setup times and alternative process plans while the goal is to minimize the total weighted tardiness and the total processing costs. Each tardy activity induces a penalty cost added to the total weighted tardiness (TWT), i.e. the first part of the objective function. Moreover, a processing cost is assigned to each activity and therefore, each activity present in the schedule increases the total processing costs (TPC) and therefore also the objective function.

2 Related Works

Abdul-Razaq et al. [2] and Madureira [3] published a survey of scheduling techniques for the single machine total weighted tardiness (STWT) problem. Recently, many authors employed evolutionary algorithms to solve the total weighted tardiness problem. Chou [4] presented an experienced learning genetic algorithm, Anghinolfi [5] used a discrete particle swarm optimization and Tasgetiren [6] published a discrete differential evolution method for solving the STWT problem. An extension of the STWT problem with many real applications is the STWT problem with controllable processing times.

R. Čapek: E-mail: capekrom@fel.cvut.cz

P. Šůcha: E-mail: suchap@fel.cvut.cz

Z. Hanzálek: E-mail: hanzalek@fel.cvut.cz

Department of Control Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Such problems deal with the situation where the nominal value of the processing times of activities (or jobs) are subject to a decision. Shorter processing times lead to a shorter schedule and more accurate meeting of due dates but the abbreviation of a processing time results in a penalty cost in the objective function. A broad survey of the problems with controllable processing times was provided by Shabtay [7]. Usually, a processing time can take an arbitrary value from the given continuous interval and only a few works ([8, 9]) have focused on the situation when a processing time has a finite set of possible integer values.

Our approach can easily model the case with the finite set of integer values of processing times for each activity where each value of processing time corresponds to a specified processing cost for an activity. Furthermore, the number of activities and precedence relations can differ among the alternative process plans. Up to our knowledge there is no work dealing with total processing costs and alternative process plans on dedicated machines and therefore we focus on such a case. In our previous work ([10]), we have focused on resource constrained project scheduling problem with positive and negative time-lags while minimizing the schedule length. In the present paper, we also use the alternative process plans but the criterion is adapted to take into account due dates and the processing costs of activities. Moreover, problem constraints are also modified such that simple precedences are assumed instead of generalized temporal relations (also called time-lags).

3 Problem Statement and Solution Method

We consider a problem of scheduling a set of n activities $\mathcal{A} = \{1 \dots n\}$ on a set of m dedicated machines $\mathcal{M} = \{M_1 \dots M_m\}$. Processing time p_j , due date d_j and the machine dedication θ_j are given for each activity. The problem can be formulated as $PD | \text{res } m11, \text{prec}, o_{ij}, \text{alts} | \sum w_j \cdot T_j + TPC$ using the well-know $\alpha|\beta|\gamma$ notation [1, 11]. More precisely, we deal with the non-preemptive scheduling where activities are constrained by the precedence relations. The term o_{ij} denotes the presence of the sequence dependent setup times and alts stands for the alternative process plans.

The goal of the scheduling is to minimize the objective function consisting of two parts. The first part $\sum w_j \cdot T_j$ represents the total weighted tardiness where $T_j = \max(0, s_j + p_j - d_j)$ stands for the tardiness of activity j , s_j is the start time and w_j is the weight of activity j . The second part TPC stands for the total processing costs. In our case $TPC = \sum c_j \cdot v_j$ where c_j is the processing cost of activity j and v_j is a binary decision variable denoting whether activity j is present in the schedule or not.

To solve the proposed model, we use evolutionary algorithms. Both discrete particle swarm optimization proposed by Anghinolfi [5] and discrete differential evolution proposed by Tasgetiren [6] algorithms proved to be effective for the STWT problem with sequence dependent setup times. The advantage of both population based methods is the utilization of an intensification phase for individuals. In this phase, the algorithm can profit from the specific features of the problem and improve individuals using local search or more complex method in each iteration of an evolution.

For our $PD | \text{res } m11, \text{prec}, o_{ij}, \text{alts} | \sum w_j \cdot T_j + TPC$ problem, we have to extend the selected method to handle alternative process plans at first. The algorithm has to return not only the sequence of activities on each machine (and consequently also start times of activities) but also the presence of activities in the schedule. Therefore, we have to extend the population based method by a functionality of selecting and rejecting of activities according to the defined alternative process plans. Furthermore, the algorithm

has to be also extended to handle different machine environment - dedicated machines instead of single machine. Precedence relations between activities are also considered. Finally, the objective function has to be modified such that it consists of two parts, i.e. total weighted tardiness and total processing costs.

4 Contribution

The main contribution of our work is in the interconnection of the scheduling with alternative process plans and the objective function taking into account both weighted tardiness and processing costs of activities. We propose a model allowing to define different ways how to complete a product while both meeting of due dates and production expenses are considered. For the given problem, we propose a population based algorithm for solving large instances. The key point of successful implementation of an evolutionary algorithm is an addition of alternative process plans into the representation of individuals and also into the evolution scheme.

References

1. Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Weglarz, J.. Scheduling Computer and Manufacturing Processes. Springer-Verlag New York, Inc.; 1996.
2. Abdul-Razaq, T., Potts, C., Van Wassenhove, L.. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics* 1990;26:235–253.
3. Madureira, A.. Meta-heuristics for the single-machine scheduling total weighted tardiness problem. In: *Proceedings of the IEEE International Symposium on Assembly and Task Planning*. 1999, p. 405–410.
4. Chou, F.D.. An experienced learning genetic algorithm to solve the single machine total weighted tardiness scheduling problem. *Expert Systems with Applications* 2009;36:3857–3865.
5. Anghinolfi, D., Paolucci, M.. A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research* 2009;193:73–85.
6. Tasgetiren, M., Pan, Q.K., Liang, Y.C.. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. *Computers and Operations Research* 2009;36:1900–1915.
7. Shabtay, D., Steiner, G.. A survey of scheduling with controllable processing times. *Discrete Applied Mathematics* 2007;155:1643–1666.
8. Chen, Z.L., Lu, Q., Tang, G.. Single machine scheduling with discretely controllable processing times. *Operations Research Letters* 1997;21:69–76.
9. Zhang, S.X., Cao, Z.G., Zhang, Y.Z.. Scheduling with discretely compressible processing times to minimize makespan. In: *The Sixth International Symposium on Operations Research and Its Applications*. 2006, p. 258–263.
10. Čapek, R., Šůcha, P., Hanzálek, Z.. Production scheduling with alternative process plans. Under review after major revision in *European Journal of Operational Research* 2011;.
11. Kellerer, H., Strusevich, V.A.. Scheduling problems for parallel dedicated machines under multiple resource constraints. *Discrete Applied Mathematics* 2003;133:45–68.

Scheduling of parallel machines with sequence-dependent batches and product incompatibilities in an automotive glass facility

Alvaro J. Lozano • Andrés L. Medaglia

The manufacturing process of automotive bullet-resistant glass involves a bending process inside an industrial furnace in small batches. The bending process capacity often limits the production line throughput. Moreover, the sequence in which batches are processed in the furnace has a considerable impact on the furnace utilization level and on the completion date for each part.

In our particular facility, owned by one of the world's largest curved bullet-resistant glass manufacturers, the bending workstation includes three furnaces with different technical characteristics. The common practice is to maintain a *buffer* with a given number of glass parts ready to be processed in any of the furnaces. The furnace operators have a predefined set of simple guidelines, which they combine with their own experience, to build and sequence *batches* attempting to simultaneously maximize the utilization level of each furnace and minimize the delay in the completion date of each part. Note that the most efficient way of loading a furnace may result in considerable delay of a given urgent part, especially, when this part has an early due date but a long processing time. Additionally, due to the *sequence-dependent* nature of the furnaces, a part with a long processing time also affects negatively the processing time of the parts scheduled after it. The processing time of a glass part is proportional to its thickness, which depends on the level of protection required in case of a bullet impact (ballistic level). Not only the processing time of a given batch is determined by the *ballistic level family* of its parts, but by the ballistic level family of the batches processed before. The batch *capacity* is defined by the physical space available in each furnace for processing multiple parts in the same compartment at the same time. Furthermore, the batch building process is restricted by a *product family incompatibility*, meaning that batches are required to group parts with the same product family, which is the same as each part's ballistic level family. There is also a *resource constraint* on the quantity of available casts for each reference; and a *furnace capability* constraint, which means that each furnace is capable of processing certain parts depending on its physical characteristics.

Our specific scheduling problem consists of scheduling n jobs (glass parts) on h parallel batch processing machines (furnaces), assigning each job to a batch, and sequencing the batches on each machine. In essence, this is a bi-objective problem in which we would like to simultaneously maximize the machines' utilization while completing the jobs within the

Alvaro J. Lozano

Centro para la Optimización y Probabilidad Aplicada (COPA), Departamento de Ingeniería Industrial, Universidad de los Andes, Bogotá, Colombia
E-mail: aj.lozano48@uniandes.edu.co

Andrés L. Medaglia

Centro para la Optimización y Probabilidad Aplicada (COPA), Departamento de Ingeniería Industrial, Universidad de los Andes, Bogotá, Colombia
E-mail: amedagli@uniandes.edu.co

required due dates. In this work we considered the offline version of the problem where all jobs are available at time zero.

There are two distinctive features that define the problem: the batch capacity and the sequence-dependent processing times. The batch size is defined by the sum of the widths of all the parts assigned to a given batch, which in turn determines if the batch fits a given machine. On the other hand, the processing time of a given batch affects the processing time of all the batches with shorter processing times that enter the machine after it (until the former batch leaves the machine). In addition, our rich scheduling problem is further characterized by three additional side constraints: (i) a product family incompatibility constraint; (ii) a resource constraint for each batch arising from a limited number of casts; and (iii) a machine capability constraint, which determines if a given job can be processed (or not) in a given machine.

According to the classical $\alpha|\beta|\gamma$ notation for scheduling problems, our particular problem is denoted by $(R_m | \text{batch, incompatible, non-identical, res, } s_{psd}, \mathcal{M}_j | C_{\max}, \Sigma T_j)$, where the symbols from left to right represent: m parallel unrelated processors, batches, incompatible products, different sized jobs, additional resources, past-sequence-dependent setup times, machine capability; and two objectives, namely, minimizing makespan (equivalent to maximizing machine utilization) and minimizing total tardiness. It is worth mentioning that our particular problem considers additional objectives for meeting practical requirements of the batches.

To some extent, our work is related to that of other researchers. Chou [1] approached the single batch-processing machine scheduling problem, minimizing makespan with a two-stage approach: first, he defined the job sequences using a genetic algorithm; then, he built the batches using dynamic programming. Mathirajan et al. [2] proposed different scheduling heuristics designed for maximizing the utilization of parallel, non-identical batch processors in the presence of dynamic job arrivals and incompatible job families. Armentano and Felizardo [3] solved the problem of scheduling jobs in uniform parallel machines with sequence-dependent setup times, minimizing the total tardiness relative to job due dates, applying a GRASP metaheuristic with a path relinking post-optimizer. To the best of our knowledge, there is no scheduling application involving all the elements present in the context of our curved bullet-resistant glass manufacturing facility.

To solve our particular real-life complex problem, we propose a two-phase heuristic: in the first phase, we build job batches for each machine; in the second phase, we sequence the batches previously allocated to each machine.

In order to build the batches in the first phase of our solution approach, we propose a four-stage mixed-integer linear programming model with a lexicographical ordering of four objectives, each one of them addressing a desirable feature of the output batches. Each objective function is optimized independently in each stage and then included in the next stage as a constraint using the best objective value found in the previous stage. This strategy follows an ε -constraint approach. Using this method, the results of earlier stages constrain the solution space of the latter stages so that the required characteristics of the batching scheme propagate throughout the solution process. In particular, the main goal of the batching process is to minimize the total number of batches. However, the optimal first-stage solution may lead to a very unbalanced load distribution for the parallel machines; thus, in the second stage, the goal is precisely load balancing. To define a consistent batch due date for the second (sequencing) phase, the third stage seeks for the due dates of the jobs in a given batch to be as similar as possible. The purpose of the fourth and last stage is to limit the number of batches with very low utilization. At the end, the solution of the fourth stage represents a batching scheme (first-phase solution) that allocates jobs to a batch, and each batch to a single machine. The attributes of the jobs in each batch (first-phase output) define the attributes of the batches (input) in the second phase.

The objective of the second phase is to find the best possible sequence of the batches (built in the first phase), subject to a batch sequence-dependent processing time and the cast availability constraint. Note that the latter constraint links the schedules of the h parallel

machines. To solve the second phase, we implemented a GRASP metaheuristic that carefully balances computing effort and solution accuracy. This procedure performs several local searches, restarting from a different randomly generated solution every time. We adapted GRASP, so that for each initial random solution a mirror solution is built, sequencing each job in the opposite way. A (double) greedy local search with best-improvement selection based on 2-exchange movements is performed at each restart (iteration) from the random original solution and the mirror solution. To implement a single objective search, we defined the total tardiness of the schedule as the objective function and introduced the utilization objective as a constraint guaranteeing a minimum level of utilization for the machines' capacity. The output of this phase is the best solution found by the GRASP metaheuristic, namely, the sequence in which the batches built in the first phase are scheduled on the parallel machines.

We tested the proposed two-phase heuristic on ten 100-job instances with real data. The relative gap to the optimal value achieved in each of the first three stages of the first phase was less than or equal to 10% for all test instances. The average execution time for the first phase was 122.3 seconds and the worst case was 388.4 seconds, which is a reasonable time for building batches in practice. The relative gap of 10% (or less) and the practical stopping criteria guarantee a high quality level of the batches according to the four objectives used in the batch building (first) phase. Figure 1 shows the attributes of the compromise solution found for one of the test instances at the end of the first phase. In the radar graph, the objectives of the compromise solution are plotted (blue square) and compared against the best possible value for each objective represented by the 100% values (inner diamond shape).

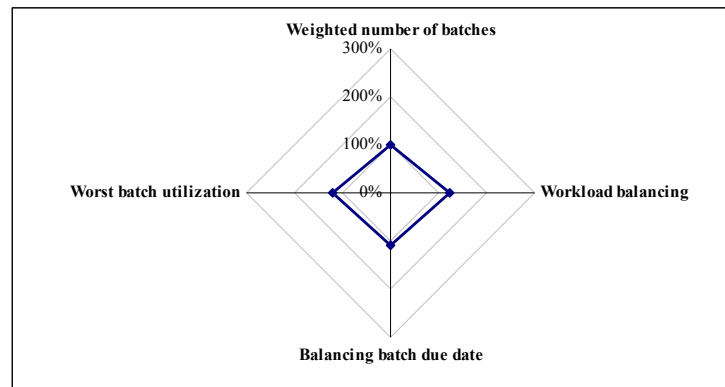


Figure 1. Illustrative example of results obtained at the end of the first phase

To evaluate the quality of the solutions found in the second phase, we estimated a lower bound for the total tardiness for each instance. The relative gap to the lower bounds was consistently below 5% and the optimal value for the second phase was achieved in 8 out of 10 cases. The execution times for the second phase were always below 335.3 seconds. Both in terms of speed and accuracy, the two-phase heuristic's performance meets the standards of the manufacturing facility that motivated our work.

Work currently underway involves the implementation of the proposed solution method in the daily production planning routine of the safety glass manufacturing facility.

The full paper is organized as follows. Section 1 introduces the real-life problem. Section 2 presents the underlying scheduling problem. Section 3 reviews the relevant literature. Section 4 describes in detail the two-phase solution approach. Section 5 presents the computational experiments with real-life data from the manufacturing facility and includes an illustrative example. Finally, Section 6 concludes the paper and outlines our future work.

References

1. Chou F.D., A joint GA+DP approach for single burn-in oven scheduling problems with makespan criterion, *International Journal of Advanced Manufacturing Technology* 35, 587–595 (2007)
2. Mathirajan M., Chandru V., Sivakumar A., Heuristic algorithms for scheduling heat-treatment furnaces of steel casting industries, *Sadhana* 32 (5), 479–500 (2007)
3. Armentano V., Felizardo M., Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory-based GRASP approach, *European Journal of Operational Research* 183, 100–114 (2007)

Single Machine Scheduling with Interfering Job Sets to Minimize Total Weighted Completion Time and Maximum Lateness

Ketan Khowala, John Fowler, Ahmet Keha, Hari Balasubramanian

1 Introduction

In the interfering job sets scheduling problems, the tradeoffs between the criteria of the sets of job often results in a set of solutions. Depending on the machine environment, performance criteria of the job sets, as well as the number of job sets, these problems can be very hard to solve.

In Khowala *et al.* [2009], we dealt with two interfering job sets on single machine with the objectives of minimizing total completion time and total number of tardy jobs for the two sets, respectively. A pseudo- polynomial algorithm was presented by Ng *et al.* [2006] for this problem under binary encoding. We presented the structure of the set of non dominated solutions and a forward SPT-EDD algorithm to generate the Pareto-Optimal frontier (also elaborated with an example). We further compared the solution quality as well as the computation times of our heuristic with MIP formulations results and with the dynamic program presented by Ng *et al.* [2006]. We demonstrated that the average percentage gap between the solution sets obtained using Forward SPT-EDD heuristic versus the Pareto optimal points was less than 1%. Computation time was under 1 second for instances with a total of 50 jobs in each job set compared to over 40 seconds using the dynamic program.

In this paper, we explore a similar problem, but with different performance criteria. In this paper, we discuss the structure of the problem and present an approach using a forward WSPT-EDD heuristic to generate a non dominated solution set.

Some of the earlier research done in this area includes, Peha [1995] who considered objectives of minimizing the weighted number of tardy jobs and the total weighted completion time on two different sets of jobs under an identical parallel machine environment. Baker and Smith [2003] formalized scheduling problems with interfering job sets. They considered a single machine problem involving the minimization of criteria including makespan, maximum

Ketan Khowala
Arizona State University
E-mail: Ketan.Khowala@asu.edu

John Fowler
Arizona State University
E-mail: John.Fowler@asu.edu

Ahmet B Keha
ExxonMobil Research and Engineering Company
E-mail: ahmet.b.keh@exxonmobil.com

Hari Balasubramanian
University of Massachusetts
E-mail: hbalasubraman@ecs.umass.edu

lateness, and total weighted completion time ($C_{\max}, L_{\max}, \sum w_j C_j$) on different sets of jobs. Agnetis et al. [2004] presented the complexity of generating non dominated solutions for a single machine as well as for shop floor scheduling problems with interfering job sets, involving objectives of minimizing total completion times, total number of tardy jobs and total weighted completion times ($\sum C_j, \sum U_j, \sum w_j C_j$). Ng et al. [2006] and Cheng et al. [2006] addressed the computational complexity of the various single machine problems with interfering job sets. Balasubramanian et al. [2008] was one of the first papers to have a heuristic based approach for generating non dominated points for the interfering job sets on the parallel machine environment. The paper proposes an iterative SPT-LPT-SPT heuristic and a bicriteria genetic algorithm for the problem by exploiting the structure of the problem.

Wan et al. [2009, 2010] dealt with two agent scheduling problems with controllable processing times, where the cost of compression is included in the objective function of the first agent. The machine environment is single or identical parallel machines. Lee et al. [2011] developed a branch-and-bound algorithm and a simulated annealing heuristic algorithm to address a two machine flow shop problem with two agents. The objectives considered were to minimize the total completion time for the first agent with no tardy jobs for the second agent.

2 Problem Description

The problem that we are investigating is denoted as $1|inter|ND(\sum w_j C_j, L_{\max})$. Clearly the notation highlights the interference between the job sets and also indicates that we are looking to find the *non-dominated* (or *Pareto optimal*) points for this problem. The interfering jobs from two disjoint sets have the objective of minimizing total weighted completion time and minimizing the maximum lateness. As stated earlier, the complexity of this problem has been established as NP Hard by Ng et al. [2006].

There are two disjoint interfering sets of jobs ξ_1 and ξ_2 with n_1 and n_2 number of jobs in each respective set. The total number of jobs that need to be scheduled is $n = (n_1 + n_2)$. We seek to minimize the total weighted completion time of the jobs in the first set ξ_1 and for the jobs in second set ξ_2 we want to minimize the maximum lateness. The processing time of the jobs in set ξ_1 and set ξ_2 are represented by p_j^1 and p_j^2 , respectively. Similarly the due dates for the jobs in the first set and the second set can be denoted by d_j^1 and d_j^2 , respectively. However, for the purpose of the objectives considered herein, due dates are only relevant for the jobs in second set.

3 Structure of the Non Dominated Solutions

The $1||\sum w_j C_j$ problem can be solved in polynomial time using the Weighted Shortest Processing Time (WSPT) rule and the $1||\sum L_{\max}$ problem can be solved in polynomial time using the Earliest Due Date (EDD) rule. However, the complexity of these performance measures with interference (two agent problem) is NP hard. Some of the properties of non-dominated solutions with interfering job sets and with these objectives are listed below.

Lemma 1: There always exists a non-delay schedule for all the strongly non dominated points on the Pareto optimal front.

Lemma 2: For all the strongly non dominated points, there exists an optimal schedule in which jobs in the job set ξ_2 are scheduled in EDD order, Baker and Smith [2003].

Lemma 3: For any non dominated point, the performance criteria ($\sum w_j C_j$) of the jobs in the job set ξ_1 with preemptive scheduling remains the same as with non preemptive scheduling, provided the jobs in job set ξ_2 which caused the preemption is scheduled before the job that got preempted in ξ_1 .

We define three subset of jobs S_1 , S_2 and S_3 . The subset of jobs S_2 and S_3 will contain all the jobs from set ξ_2 arranged in EDD order. All the jobs in subset S_2 will be scheduled together and the last job (j_*) in subset S_2 will be the job defining the $L_{\max}$ criterion for jobs in set ξ_2 . All the jobs in subset S_3 are jobs from set ξ_2 that are scheduled after the $L_{\max}$ job in the EDD order. The jobs in S_3 may have some slack and could be delayed without impacting the $L_{\max}$ value for that non dominated point. The jobs in subset S_1 are all the jobs from set ξ_1 . The jobs in subset S_1 may not follow the WSPT order (dominance criteria may apply). This structure of the non dominated points is presented in the picture below.

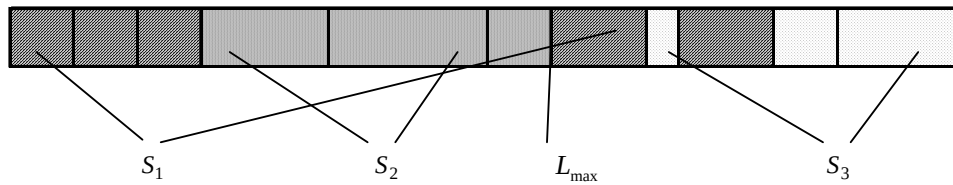


Figure 1: Structure of the non dominated point with total weighted completion time and maximum lateness as performance criteria

4 Forward WSPT-EDD Heuristic

The Forward WSPT–EDD heuristic can be summarized by the following steps:

Step1: Arrange the jobs in ξ_2 in EDD order starting at time zero.

Step2: Find the job j_* from ξ_2 which has the $L_{\max}$ value. This job will divide the jobs in the second set into S_2 and S_3 .

Step3: All the jobs in S_2 will occur in a block with the last job being j_* . The jobs that are scheduled after this job j_* (jobs in S_3) can be moved further in the time horizon to take advantage of the slack with respect to their completion time and $L_{\max}$ value. This will create an opportunity for improvement in the performance criterion of jobs in S_1 without impacting the $L_{\max}$ value.

Step4: Schedule the jobs in S_1 according to WSPT (and assuming preemption is allowed) in between jobs from S_2 and S_3 . Correct for preemption of jobs in S_1 by moving the jobs in sets S_2 and S_3 ahead in the time horizon.

Step5: Apply pair wise swap for jobs in S_1 to see if there is any improvement in the $\sum w_j C_j$ value.

Step6: Repeat steps 3 through step 5 every time incrementing the $L_{\max}$ value by one unit, until all the jobs in S_1 are scheduled at the beginning of the time horizon.

Currently, we are also working on some dominance rules that can be applied to the jobs in S_1 after initial WSPT schedule. Note that the WSPT rule is not always optimal for jobs in set ξ_1 with interference (Baker & Smith [2003]). These dominance rules should help improve the value for $\sum w_j C_j$ in the final schedule with interference. The WSPT order for the jobs in set S_1 can potentially be affected whenever any job from set S_3 is moved ahead in time to avoid preemption of jobs in S_1 . Whenever any job in S_1 is preempted by jobs in S_3 (and causing jobs from S_3 to be moved ahead to avoid preemption), there could be potential improvement in $\sum w_j C_j$ with swaps between this S_1 job and the subsequent S_1 job in the schedule.

5 Computational Experiments & Future Research

Currently, we are working on improving the proposed forward WSPT-EDD heuristic as well as developing the dominance rules. We will be using a similar framework as used in Khowala *et al.* [2009] to compare the results from this approach to the solution sets obtained from the MIP formulations for this problem. For larger instances of the problem, we will establish a lower bound and compare the results from the two approaches in terms of solution quality and computational efficiency. Further the findings from Khowala *et al.* [2009] as well as this current research could be used to look into single machine two agent problems with total weighted completion time and number of tardy jobs problem which is also a NP hard problem. Our further investigations, results and continued research will be presented during the conference.

References

1. Agnetis, Allesandro; Mirchandani, Pitu B.; Pacciarelli, Dario and Pacifici, Andrea; "Scheduling problems with two competing agents", *Operations Research*, 52(2), 229-242, 2004.
2. Baker, Kenneth R. and Smith, J. Cole; "A multiple criterion model for machine scheduling", *Journal of Scheduling*, 6, 7-16, 2003.
3. Balasubramanian H.; Fowler J.; Keha A. and Pfund M.; "Scheduling interfering job sets on parallel machines", *European Journal of Operational Research*, 199(1), pp 55-67, 2009.

4. Cheng, T.C.E.; Ng, C.T. and Yuan, J.J; "Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs", *Theoretical Computer Science*, 362, pp 273-281, 2006.
5. Jazzkiewicz, A; "Evaluation of Multiple Objective Metaheuristics", *Lecture Notes in Economics and Mathematical Systems*, 535, pp 65-89, 2003
6. Khowala, K.; Fowler, J.; Keha, A. and Balasubramanian, H; "Single machine scheduling with interfering job sets", *Proceedings of the Multidisciplinary International Scheduling: Theory and Applications Conference (MISTA 2009)*, pp 357-365, Aug 2009.
7. Lee, W.-C.; Chen, S.-K.; Chen, C.-W. and Wu, C.-C; "A two-machine flowshop problem with two agents", *Computers & Operations Research*, 38, pp 98-104, 2011.
8. Ng, C.T.; Cheng, T.C.E. and Yuan, J.J; "A note on the complexity of the problem of two agent scheduling on a single machine", *Journal of Combinatorial Optimization*, 12, 387-394, 2006.
9. Peha, J.; "Heterogeneous-criteria scheduling: minimizing weighted number of tardy jobs and weighted completion time", *Journal of Computers and Operations Research*, 22(10), 1089-1100, 1995.
10. Wan, G.; Leung, J. Y.-T. and Pinedo M; "Competitive agent scheduling with Controllable Processing Times", *Proceedings of the Multidisciplinary International Scheduling: Theory and Applications Conference (MISTA 2007)*, pp 514-522, Aug 2009.
11. Wan, G.; Vakati, S. R.; Leung, J. Y.-T. and Pinedo M; "Scheduling two agent with controllable processing times", *European Journal of Operational Research*, 205(3), pp 528-539, 2010.

Investigating the boundary between scheduling and dispatching

Michele E. Pfund • John W. Fowler

1 Introduction

This research presents the case of a Japanese Printed Wiring Board Manufacturer's drilling operation, which is a group of unrelated parallel machines. The manufacturer wishes to develop an algorithm that can minimize makespan, number late, total overtime, average finishing time, and utilization and find a solution in under 10 seconds. We develop a Lagrangian relaxation formulation and compare it dispatching methods. The success of the approach is evaluated using a simulation model to evaluate the performance over a number of sequential schedules and differing levels of processing variability.

2 System Description

The process of printed wiring board manufacturing is comprised of several sequential steps. Within our manufacturing system, the drilling area is the bottleneck and is comprised of 14 numerical control (NC) machines that are unrelated as each machine has a unique processing time for each of the factory's different types of lots. In addition, each machine may not be capable of processing all lots. However, these machines are highly flexible so that switching from one product to another requires minimal setup time.

Lots arrive to this area by an automated guided vehicle (AGV) every T hours (referred to as the release interval). In this research, T is three hours and we assume that the number of lots that arrive on the AGV is normally distributed with a mean of 30 and standard deviation of 1.5. After delivering the product, the AGV retrieves the parts made by the drilling operation during the previous release interval and delivers them to the copper plating station. If any lots delivered in the previous delivery are not complete, they are considered to be late and must be transported to the next station by hand. The amount of time past T is considered overtime.

3 Solution Methodologies

Michele Pfund
Arizona State University – Department of Supply Chain Management
E-mail: Michele.pfund@asu.edu

John Fowler
Arizona State University – School of Computing, Informatics and Decision Systems
Engineering
E-mail: John.Fowler@asu.edu

LRH, Lagrangian Relaxation Heuristic (scheduling approach)

LRH is a two-phase heuristic / optimization scheduling approach. In the first phase, a heuristic finds critical jobs and assigns them to the machines with the shortest total assigned processing time. These preferential assignments are then integrated into the objective function of an integer program through Lagrangian relaxation. (Details regarding the heuristic procedure can be found in Yu et al., 2002). The remaining constraints of the integer program enforce scheduling feasibility constraints and have a unimodular structure. Thus, the resulting optimization routine can be solved in polynomial time.

$$\text{Minimize } \sum_{i=1}^m \sum_{j=1}^n \alpha_{ij} \sum_{k=1}^n k(p_{ij} + O_i)x_{ijk} + \sum_{\{abk_1\} \in \Lambda} \lambda(1 - x_{abk_1}) + \sum_{\{ck_2\} \in \Pi} \pi \sum_{j=1}^n \sum_{k=k_2}^n x_{jk_2}$$

subject to:

$$\sum_{j=1}^n x_{ijk} \leq 1 \quad (i = 1, 2, \dots, m)(k = 1, 2, \dots, n)$$

$$\sum_{i=1}^m \sum_{j=1}^n x_{ijk} = 1 \quad (j = 1, 2, \dots, n)$$

$$x_{ijk} = 0 \text{ or } 1 \quad (\forall i, j, k)$$

where i is the machine index, j is the job index, and k refers to the position (from the last) of the job on a machine. p_{ij} is the processing time of job j on machine i and O_i refers to the amount of overtime on a machine.

LHR+, Lagrangian Relaxation Improved (scheduling approach)

An improved version of LRH was created by eliminating the first phrase of LRH and incorporating total overtime constraints for each machine through Lagrangian relaxation. The constraints remain the same as LRH but the objective function changes and a subgradient optimization procedure employing a desirability function finds the values of λ_i .

$$\text{Minimize } \sum_{i=1}^m \sum_{j=1}^n \sum_{k=1}^n k(p_{ij} + O_i)x_{ijk} + \sum_{i=1}^m \lambda_i \left(\sum_{j=1}^n \sum_{k=1}^n p_{ij}x_{ijk} + O_i - T \right)$$

Ibarra and Kim Algorithm A (scheduling approach)

Schedule jobs in first-come-first server order to the machine that would have the earliest completion time. See Ibarra and Kim (1977) for details.

ASDN (dispatching approach)

Select jobs with the lowest ASDN index using average processing time of a job j on all feasible machines (A_j), standard deviation of processing times of job j across all feasible machines (SD_j), and the number of feasible machines job j can be processed upon (M_j) divided by the total number of machines ($|M|$).

$$ASDN_j = \frac{e^{(p_{ij} - A_j)} * \frac{M_j}{|M|}}{SD_j}$$

HASDN (hybrid scheduling / dispatching approach)

Schedule critical jobs using 1st Phase LRH. Dispatch remaining jobs using ASDN.

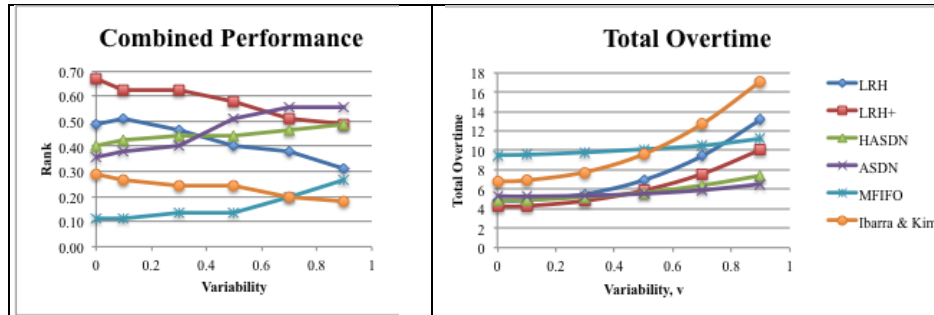
MFIFO (dispatching approach)

Chooses the job which is feasible for a given machine and arrived first to the queue.

4 Results

20 replications of 160 schedules (1 month of operation) were performed to assess the long-term performance of the system when variability increased. Variability was restricted to processing times which were modeled using a symmetric triangular distribution with $\min = (1-v) \cdot \text{mean}$, $\text{mode} = \text{mean}$, and $\max = (1+v) \cdot \text{mean}$ where v is a value between 0 and 1.

Detailed results for the makespan, number late, total overtime, average machine finishing time, and utilization were collected for each replication. Upon completion of all simulation replications, the performance of each methodology was assessed using a normalized ranking function where rank=1 indicates the method performed best across all objectives and rank = 1/6 indicates the method was ranked last across all performance measures. Abbreviated results are shown in the figures below:



These graphs highlight several important relationships between scheduling and dispatching when conditions of variability are present.

- As variability increases, scheduling rules decrease significantly in performance. We can see this in the total overtime plot. Note that all scheduling algorithms (LRH, LRH+, Ibarra and Kim) show a similar rate of deterioration.
- As variability increases, dispatching rules decrease in performance but not significantly. ASDN and MFIFO are both dispatching rules and though we see increases in total overtime, the amount of increase is significantly less than the scheduling algorithms.
- The better the scheduling rule, the more robust the solution is to processing variability. We see this in the comparison of LRH and LRH+'s performance in the combined rank graph. When there is no variability, we can see that LRH+'s performance is superior. As variability increases, the indifference point between scheduling and dispatching is about 0.4 for LRH but is about 0.7 for LRH+.
- Dispatching rules can dominate scheduling rules. Ibarra and Kim is designed for sum of completion times only, not all 6 metrics. Reviewing the chart, we observe that ASDN dominates Ibarra and Kim even at low variability conditions.
- FIFO is a competitor to poorly performing scheduling rules. The Ibarra & Kim rule is not designed for a multi-objective environment. Though it performs very well for sum of completion times, its poor performance with other metrics cause it to be dominated by FIFO as variability increases.

References

1. Ibarra, O. and C. Kim, "Heuristic Algorithms for Scheduling Independent Tasks of Nonidentical Processors," *Journal of the Association for Computing Machinery*, **24**, 280-289, 1977.
2. Yu, L., Shih, H.M., Pfund, M., Carlyle, W.M., and Fowler, J.W., "Scheduling of Unrelated Parallel Machines: An Application to PWB Manufacturing", *IIE Transactions on Scheduling and Logistics*, **34**(11), 921-931, 2002.

A Parallel Approach for a Nurse Rerostering Problem on the GPU

Bäumelt Zdeněk · Šůcha Přemysl · Hanzálek Zdeněk

1 Introduction

This paper deals with a *Nurse Rerostering Problem* (NRRP) that is handled every day in hospitals in case of some unpredicted circumstances (e.g. sick leaves of the nurses). Consequently, it is necessary to move the shift of the absent nurse to someone else in the roster. However, NRRP cannot be usually solved by the same methods used for another well known *Nurse Rostering Problem* (NRP), since each change of the original roster may lead to collisions with the nurses' already planned free-time activities. Therefore, the main objective is to solve NRRP respecting the minimum changes in the original roster. Moreover, the roster has to satisfy a given set of constraints to preserve the structure of the original roster.

This paper proposes a parallel metaheuristic approach for NRRP accelerated on the *Graphics Processing Unit* (GPU) that offers a new way to tackle this problem. In general, this platform allows us to employ massively parallel processors to achieve the speedups up to hundred times (according to the handled problem) in comparison with the approaches implemented on CPU.

2 Problem Statement and Related Works

The NRRP discussed in this paper is defined as follows. Let E be a *set of the nurses*, D be a *set of the days* and S be a *set of the shifts*. Subsequently, let R^0 be a binary matrix standing for an *original roster* (the NRP output) of a size $E \times D \times S$, i.e. $R_{ijs} = 1$ denotes that shift s is assigned to nurse i on day j . Similarly, a *modified roster* is defined by R . Afterwards, let A be a binary *matrix of absence* of a size $E \times D \times S$, i.e. the fact that nurse i is not able to work shift s on day j is expressed by $A_{ijs} = 1$. Furthermore, let Δ_{max}^R be a *maximum number of modifications* in R according to R^0 . Consequently, the NRRP is given by a triplet $\{R^0, A, \Delta_{max}^R\}$.

We consider the NRRP containing the shifts {early, late, night, free (day-off), holiday (fixed day-off)}. In order to determine whether shift s_1 can be followed by shift s_2 or not, we define a binary matrix SP of a size $S \times S$. Furthermore, *requested shifts* are stated by a binary matrix RS of a size $S \times D$, i.e. RS_{sj} represents the count of the requested shifts of type s on day j . Finally, one has to distinguish which parts of the roster are fixed and which can be modified. For this purpose, let F be a binary *matrix of the fixation* having a size $E \times D$. We take into account the following assumptions of NRRP: a) all shifts have the same length equal to 8 hours; b) all nurses have the same grades, i.e. all shifts can be assigned to them; c) all nurses have the same workloads.

Z. Bäumelt · P. Šůcha · Z. Hanzálek

Department of Control Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Technická 2, 166 27, Prague 6, Czech Republic
E-mail: baumezde@fel.cvut.cz, suchap@fel.cvut.cz, hanzalek@fel.cvut.cz

The objective of the NRRP is to modify R^0 in order to find R that satisfies following hard constraints. The constraint (hc_1) assumes that nurse i is assigned to exactly one shift per day j , i.e. $\sum_s R_{ijs} = 1$ must hold for $\forall i \in E, \forall j \in D$. Let Δ^R be a number of modifications in R . Thereafter, the constraint (hc_2) guarantees that $\Delta^R = 1/2 \cdot \sum_{i,j,s} |R^0 - R| \leq \Delta_{max}^R$. The constraint (hc_3) ensures that all fixed shifts in R cannot be changed, i.e. $\forall F_{ij} = 1 \Rightarrow R_{ijs} = R_{ijs}^0, \forall i \in E, \forall j \in D, \forall s \in S$. Moreover, the available nurses have to be taken into account only (hc_4), i.e. $\forall A_{ijs} = 1 \Rightarrow R_{ijs} = 0, \forall i \in E, \forall j \in D, \forall s \in S$. The constraint (hc_5) avoids the forbidden shift precedences, i.e. $\forall SP_{s_1 s_2} = 0 \Rightarrow R_{ijs_1} \wedge R_{i,j+1,s_2} = 0, \forall i \in E, \forall j < |D|, \forall s_1, s_2 \in S$. The last hard constraint (hc_6) expresses that the coverage of shifts has to be met, i.e. $RS_{sj} = \sum_i R_{ijs}, \forall j \in D, \forall s \in S$.

Naturally, NRRP contains the same soft constraints as NRP to keep the original roster. Let Z^Δ be a *multiobjective function*, given by a linear combination of the weighted penalizations of the particular soft constraints, that is minimized. The total workload balancing among nurses (sc_1) is incorporated in Z^Δ by $\sum_i \left| \sum_{j,s} R_{ijs} - \overline{W} \right|$, where $\overline{W}$ is an *average workload* equal to $1/|E| \cdot \sum_{s,j} RS_{sj}$. The constraint (sc_2) takes into account the maximum number of consecutive days-on. Finally, the count of modifications Δ^R is minimized by (sc_3) in order to keep R as close to R^0 as possible.

Most of the approaches dealing with NRP are summarized in [1]. Unfortunately, NRRP receives a much less attention than NRP, though NRRP has to be solved as often as NRP. Several approaches are presented in papers by Moz and Pato. The first paper [2] models NRRP by an integer multicommodity flow represented by a multi-level acyclic network. Another approach based on a genetic algorithm was described in more variations in papers [3] and [4] (contains the proof that NRRP is NP-complete based on the polynomial-time reduction to another well-known NP-complete problem, the three-dimensional matching problem). Different strategies of evolutionary algorithms were discussed in [5] where the results of Moz and Pato were outperformed. Another approach for NRRP based on a recursive search algorithm is presented in [6].

We have observed from our preliminary results that the time complexity of the NRRP approach is crucial. We implemented a recursive search algorithm with the bounded depth of the recursion ($\sim \Delta_{max}^R$) on the CPU. The computational time for instances with $|E| = 50, \Delta_{max}^R = 4$ reached dozens of seconds, while for instances with the same parameters and $\Delta_{max}^R = 8$ it increased to hundreds of seconds, which was the greatest disadvantage of the approach. Moreover, this approach was memory consuming, too. Therefore, our intention is to propose the parallel approach for NRRP designed for GPU without these drawbacks.

In general, the success of the GPU technology lies in a few preconditions. Firstly, an algorithm has to be processable in terms of the single instruction multiple data parallelization [7]. Secondly, the algorithmic representation of the problem has to consider the limitations of the GPU (a limited memory). Finally, it is necessary to use the computational power in an efficient way to achieve the expected speedups of the parallel algorithm on GPU in comparison with CPU approaches.

3 Proposed Approach

We have decided to solve NRRP by a parallel genetic algorithm (GA) implemented on the GPU platform based on an island model of the population (presented in [8]). The outline of the GA is following. An *initial population* $\mathcal{P}_0$ containing *individuals* $\mathcal{I}$ is generated. Subsequently, an evaluation of the current population, a selection and

a recombination process are performed repeatedly. The stop condition is met when the total count of iterations is reached or the value of Z^Δ is decreased to 0.

Each individual $\mathcal{I}$ of a population $\mathcal{P}$ is defined by $\mathcal{I} = \{M^R, T^R\}$ as follows.

Let $M^R = \{(i_1, j_1, s_{n_1}), (i_2, j_2, s_{n_2}), \dots\}$ be a *list of modifications* in R of a length Δ_{max}^R . Each element of M^R corresponds to one *modification* in R with respect to R^0 , i.e. a *new shift* $s_n \neq s$ is assigned to nurse i on day j . Furthermore, let $T^R = \{(i_1, i'_1, j_1), (i_2, i'_2, j_2), \dots\}$ be a *list of exchanges* in R of a length $\lceil 1/2 \cdot \Delta_{max}^R \rceil$. In this case, each triplet of T^R represents one *exchange* of shifts in R between nurses i and i' on day j . This representation is redundant from the data point of view, but its benefit lies in the efficient performance of GA on GPU described in the next paragraph.

The T^R representation is more appropriate for the generation of the initial population $\mathcal{P}_0$. Each individual can be created by filling of the T^R randomly under the policy of some relations in T^R , e.g. for each 2 following exchanges t and $t+1$ in T^R holds that $i_t = i_{t+1} \vee i_t = i'_{t+1} \vee i'_t = i_{t+1} \vee i'_t = i'_{t+1}$. Similarly, the T^R representation is more efficient for the selection and the recombination process. On the contrary, the M^R representation is more effective for the evaluation of all $\mathcal{I}$ in $\mathcal{P}$ and therefore, T^R has to be mirrored to M^R to evaluate the current population. The evaluation is performed over the nurses and days with the modifications only.

4 Contribution

We have proposed the approach for the nurse rostering problem (NRRP) based on the genetic algorithm for the GPU platform. To the best of our knowledge, there is no paper tackling any problem like NRRP with more ‘difficult’ representation on the GPU, only the problems with very simple permutation encoding like a knapsack problem or flow shop scheduling were handled. Therefore, our contribution lies in the novel representation of the NRRP and the proposed approach for the GPU. Our objective is to implement the proposed genetic algorithm on the GPU and to compare the results of the GPU approach with our already implemented CPU solution.

References

1. Burke, E.K., De Causmaecker, P., Vanden Berghe, G., Van Landeghem, H.. The state of the art of nurse rostering. *Journal of Scheduling* 2004;7:441–499.
2. Moz, M., Pato, M.V.. Solving the problem of rostering nurse schedules with hard constraints: New multicommodity flow models. *Annals OR* 2004;128(1–4):179–197.
3. Pato, M.V., Moz, M.. Solving a bi-objective nurse rostering problem by using a utopic pareto genetic heuristic. *Journal of Heuristics* 2008;14:359–374.
4. Moz, M., Pato, M.V.. A genetic algorithm approach to a nurse rostering problem. *Computers & OR* 2007;34(3):667–691.
5. Maenhout, B., Vanhoucke, M.. An evolutionary approach for the nurse rostering problem. *Computers & Operations Research* 2010;In Press, Corrected Proof.
6. Kitada, M., Morizawa, K., Nagasawa, H.. Recursive search approach in nurse rostering following a sudden absence of nurses. In: *Proceedings of the 10th Asia Pacific Industrial Engineering & Management Systems*. 2009, p. 1219–1227.
7. Talbi, E.G.. *Parallel Combinatorial Optimization*. Wiley-Interscience; 2006.
8. Luong, T.V., Melab, N., Talbi, E.G.. Gpu-based island model for evolutionary algorithms. In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. GECCO '10; 2010, p. 1089–1096.

Pareto Optimal Negotiation for Nurse Rostering Problems

Stefaan Haspeslagh · Patrick De Causmaecker

1 Introduction

The nurse rostering problem has recently been subject of intense study [3,8,12,2,7,1]. Currently, most techniques in literature focus on constructing rosters for individual wards. This research focusses on the design and implementation of a decision support system for the distribution of workload and personnel between the wards of a hospital. Several authors discuss sharing resources between agents in practical settings. E.g. [9,4,11,10] design auction based systems to allocate resources for loading and discharging vessels in a container terminal. These contributions typically implement one or more auction systems and compare their efficiency and outcome to a central method. This method allows for a comparative study on the relative performance of a particular auction in comparison with central approaches or alternative negotiation protocols. However, such analysis never gives information about the quality of the negotiation method itself. E.g. Is the outcome of the negotiation process optimal? How can we be really sure that a certain negotiation algorithm outperforms another if we are uncertain about the optimality of the result? We apply the multi-agent multi-issue negotiation method of [14] to nurse rostering. The outcome of our scheme is Pareto-optimal, the agents can have incomplete information about their opponents' preferences and the agents utility functions can be non-linear. Section 2 elaborates on our model. In section 3 we identify the issues within our model. Section 4 describes the application. More information about the negotiation protocol itself can be found in [14].

Stefaan Haspeslagh

K.U.Leuven, Science & Technologie KULAK, Department of Computer Science, CODeS research group (Member of ITEC-IBBT-KULeuven), E. Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: stefaan.haspeslagh@kuleuven-kortrijk.be

Patrick De Causmaecker

K.U.Leuven, Science & Technologie KULAK, Department of Computer Science, CODeS research group (Member of ITEC-IBBT-KULeuven), E. Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: patrick.decausmaecker@kuleuven-kortrijk.be

2 Model

We consider a hospital with n nurses, w wards and s shifts to be covered. Each ward w_i has a preferred workload s_i of shifts to cover and staffing n_i of nurses. Each ward can operate within some boundaries, e.g. if more shifts need to be covered, then more nurses are required. Using the proposed negotiation protocol, the wards distribute the n nurses and s shifts among themselves in order to ensure the good operation of their tasks. Each ward w_i is thus a self-interested agent trying to perform it's task at minimum cost (implemented as a penalty function taking personal rules and preferences into account¹).

3 Issues for Pareto optimal negotiation

The issues we consider are availability of nurses and workload (the number of shifts to be covered by a ward). For each issue we find a region where negotiation is useful (e.g. the outcome of the negotiation mechanism will be accepted by the agents). Also, in order to ensure Pareto-optimality, we have to search for nurse rostering problem instances where the objective function resembles a concave tendency within some region. We used instances similar to those of the First International Nurse Rostering Competition 2010 (INCR2010)[8]. For each instance, we varied the number of personal available and the workload. We solved the generated instances using the winning algorithm of the INRC2010[13]. We plotted the value of the objective function in terms of increasing availability of nurses and increasing workload. Some results can be found in figure 1 and 2.

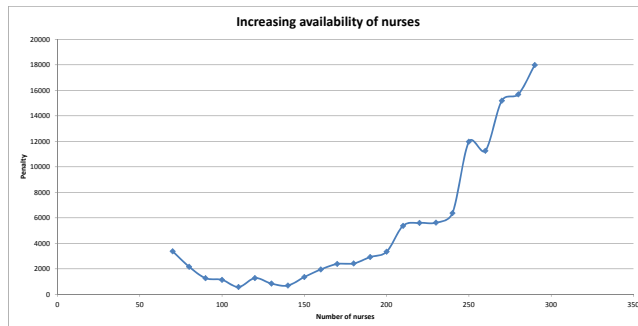


Fig. 1: Increasing availability of nurses

We can clearly identify the regions where negotiation is useful. E.g. figure 1 shows a concave tendency and acceptable penalty between 70 and 200 nurses available, thus the ward has a clear preference between possible offers in within this region. Analogously, figure 2 shows a similar tendency for increasing workload when the number of available nurses is fixed. Figure 3 shows the objective function within the feasible region.

Next, we translate this curve to a utility curve: utility 1 corresponds with minimal cost (lowest penalty), utility 0 at the borders. The result is plotted in figure 4. For each

¹ an exact definition of the objective function can be found in [8]

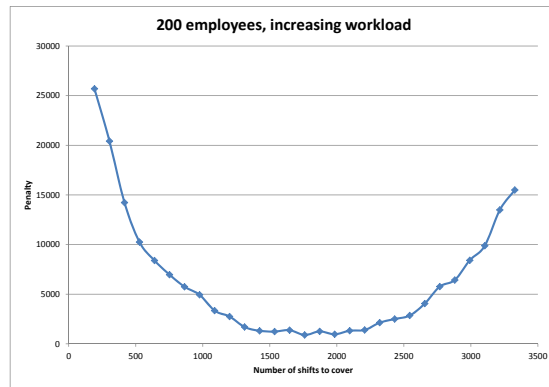


Fig. 2: Increasing workload

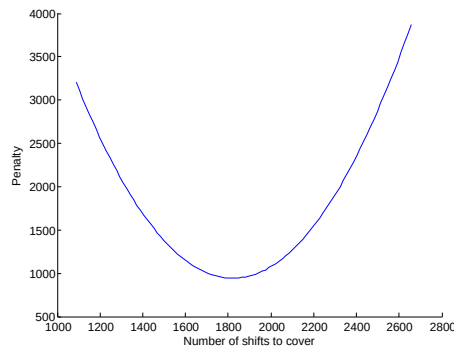


Fig. 3: Penalty in feasible region for 200 employees

generated instance, we calculate the utility curves. We then can identify indifference curves for the ward: combinations of number of employees and shifts to cover that result in the same utility level. For example, the green curve on figure 5 resembles utility level 1, the blue curves utility level 0.5.

4 Application

The model and computed dependencies in figure 5 are set to use in the framework of [14] to support the bidding processes in the wards. In the presentation we will describe these processes in detail and show computational results. Furthermore, we will speculate on the significance of such issue-characteristics for the higher levels of planning and decision making in a hospital hereby following up on [5] and the reference models for timetabling and rostering in [6].

Acknowledgements Work supported in part by project IWT 060837 DICOMAS

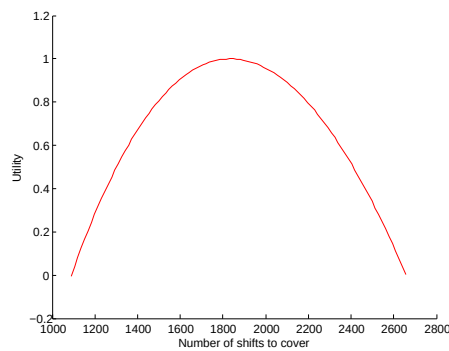


Fig. 4: Utility in feasible region for 200 employees

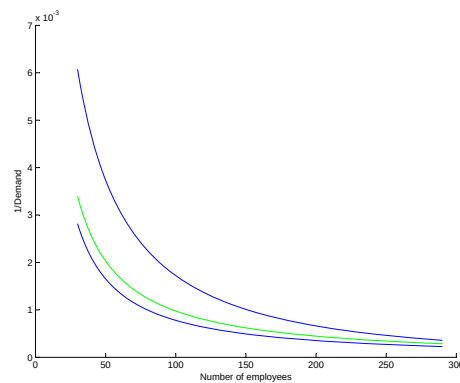


Fig. 5: Indifference curves

References

1. Beddoe, G., Petrovic, S., Li, J.: A hybrid metaheuristic case-based reasoning system for-nurserostering. *Journal of Scheduling* **12**, 99–119 (2009)
2. Bilgin, B., De Causmaecker, P., Rossie, B., Vanden Berghe, G.: Local search neighbourhoods for dealing with a novel nurse rostering model. *Annals of Operations Research* (2011)
3. Burke, E.K., De Causmaecker, P., Berghe, G.V., Van Landeghem, H.: The state of the art of nurse rostering. *Journal of Scheduling* **7**, 441–499 (2004)
4. Chen, H., Lau, H.C.: Decentralized resource allocation and scheduling via walrasian auctions with negotiable agents. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on* **2**, 356–360 (2010)
5. De Causmaecker, P., Vanden Berghe, G.: Categorisation of nurse rostering problems. *Journal of Scheduling* (2011). Accepted
6. De Causmaecker, P., Vanden Berghe, G.: Towards a reference model for timetabling and rostering. *Annals of Operations Research* (2011). Accepted
7. Goodman, M., Dowsland, K., Thompson, J.: A grasp-knapsack hybrid for a nurse-scheduling problem. *Journal of Heuristics* **15**, 351–379 (2009)
8. Haspeslagh, S., De Causmaecker, P., Stølevik, M., Schaerf, A.: First international nurse rostering competition 2010. Tech. rep., K.U.Leuven Campus Kortrijk (2010). [Http://www.kuleuven-kortrijk.be/codes/](http://www.kuleuven-kortrijk.be/codes/)
9. Lau, H.C., Cheng, S.F., Leong, T.Y., Park, J.H., Zhao, Z.: Multi-period combinatorial auction mechanism for distributed resource allocation and scheduling. In: *Proceedings of*

- the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT '07, pp. 407–411. IEEE Computer Society, Washington, DC, USA (2007)
10. Ostwald, J., Lesser, V., Abdallah, S.: Combinatorial Auction for Resource Allocation in a Distributed Sensor Network. The 26th IEEE International Real-Time Systems Symposium, (RTSS'05) pp. 266–274 (2005)
 11. Parkes, D.C., Ungar, L.H.: An auction-based method for decentralized train scheduling. In: Proceedings of the fifth international conference on Autonomous agents, AGENTS '01, pp. 43–50. ACM, New York, NY, USA (2001)
 12. Petrovic, S., Vanden Berghe, G.: A comparison of two approaches to nurse rostering problems. *Annals of Operations Research* (2010)
 13. Valouxis, C., Gogos, C., Goulas, G., Alefragis, P., Housos, E.: A systematic two phase approach for the nurse rostering problem. URL <http://www.kuleuven-kortrijk.be/nrpcompetition/competitor-ranking>
 14. Wu, M., de Weerd, M.M., La Poutre, J.A.: Efficient Methods For Multi-Agent Multi-Issue Negotiation: Allocating Resources. In: J. Yang, M. Yokoo, T. Ito, Z. Jin, P. Scerri (eds.) *Principles of Practice in Multi-Agent Systems*, pp. 97 – 112. Springer (2009)

An Exact and Heuristic Hybrid for Solving a Medical Student Scheduling Problem

Jonathan Thompson • Kathryn Dowsland

1 Introduction

This paper investigates the problem of allocating medical students to clinical specialties over a number of time periods. The objective is to ensure that all students have a broad range of experiences during their training while ensuring each student covers each specialty exactly once. A series of GRASP implementations are compared, each having the same improvement phase but different construction phases. These vary the balance between finding feasible solutions (with respect to the hard constraints) and seeking high quality solutions (with respect to the soft constraints). This paper addresses the following issues:

- The goals of solution quality and feasibility are in conflict, so a construction designed to produce high quality solutions may not be able to guarantee feasibility. Is it preferable to allow the construction to focus on feasibility and let the improvement phase concentrate on solution quality?
- As solution quality is a function of the violated soft constraints, we observe that the construction phase may complete a large part of the construction before any violations occur. Is it beneficial to replace the traditional myopic approach with a look-ahead function, or larger construction steps?
- The medical student scheduling problem considered here has a high degree of symmetry caused by homogenous events, students etc. This may mean there may be several equivalent but different solutions which may cause feedback mechanisms to return confusing information. Is there an effective way in this scenario of ensuring all the useful work done in the construction phase is not lost when considering further constructions?

Real data from a large UK medical school is used for testing and empirical results will be presented.

Jonathan Thompson
Cardiff University, UK
E-mail: ThompsonJM1@cardiff.ac.uk

Kathryn Dowsland
Gower Optimal Algorithms Limited
E-mail: k.a.dowsland@btconnect.com

2 The Medical School Scheduling Problem

During their training, medical students are required to complete several years of clinical experience, which involves completing placements in different hospitals and covering a number of specialties. The particular variant of the problem addressed here originates from Cardiff Medical School in the UK. The first clinical year is split into five time-slots, with each student studying one of five specialties, or modules, in each. In the first time-slot, all students study an introductory module; they then complete the other four exactly once in any order in time-slots 2-5. Several hospitals each provide a number of 'firms', each led by one or more consultants and offering a given number of places during each time-slot. The introductory module is offered across the board; each hospital offers a number of firms designed for this purpose and the total number of places is pre-set to be equal to the number of students. Subsequent modules are only offered by firms associated with the relevant specialty. The number of places available in each firm, the capacity, is constant over all four time-slots. For a solution to be feasible, each of the following hard constraints must be satisfied:

- C1: Each student must study the introductory module in time period 1.
- C2: Each student must study the remaining four specialties exactly once in time periods 2-5.
- C3: The number of students placed in each firm in each time period must not exceed the capacity of that firm.

It is also desirable for students to get as broad a range of experience as possible during their placements. This was originally defined by 2 further soft constraints:

- O1: Each student should not be allocated to the same consultant more than once.
- O2: Each student should not be allocated to the same hospital more than once.
- O3: Each pair of students should not be allocated to the same firm at the same time more than once.

In practice O1 is more important than O2 which in turn is more important than O3 and after some experimentation and consultation with the medical school relative weights of 50:10:1 were agreed. Violations are measured pair-wise. For example if a student studies under the same consultant on three occasions this involves 3 pair-wise violations: occasion 1 with occasion 2, occasion 2 with occasion 3, and occasion 3 with occasion 1. More generally if a student studies under the same consultant k times then the number of violations is $k(k-1)/2$. Violations of O2 and O3 are defined in the same way. We will refer to the penalty cost associated with violations of O1, O2, O3 as the consultant cost, hospital cost and student cost respectively. The problem is therefore one of minimising the weighted sum of the soft constraint violations subject to satisfying all the hard constraints.

Problems involving scheduling in medical schools have been tackled by other researchers (Franz and Miller 1993; Pentico 1980; Pentico 2007), however the problem details reported differ considerably from the variant of the problem tackled here. A previous attempt to solve the problem (Fuller 1998) used a random step-by-step construction to build a zero cost solution for as many students as possible, and then an IP model to allocate the remaining students at

minimum hospital and consultant cost. Several runs were carried out and the solution with the lowest total cost solution was selected as the best.

3 GRASP for the Medical School Scheduling Phase

During the construction phase, we must select a time slot, a student and a firm. In all cases, we complete time slot t before starting timeslot $t+1$. Our first set of constructions allocate a single student to a single firm in each step. They use a variety of score functions based on a feasibility score, which biases the choice of firm towards firms with the most remaining space left in order to ensure there is likely to be space left in all specialties for as long as possible. They also use an optimality score which biases the constructions towards solutions that minimize the secondary objectives. We consider incorporating a look-ahead function which compares the best scoring allocation for each student with their second best allocation. Priority is given to those with the largest such score, as not allocating them early is likely to lead to high costs later.

We also demonstrate that we can ensure that feasibility is maintained during the construction phase by modeling the problem as a network flow problem.

We also consider constructions that produce a schedule for every student simultaneously during a single time period. This can be done by extending the network flow model to actually allocate students to firms, though soft constraint O3 cannot be directly incorporated into the model. We add a random element to our solution algorithm to minimize the increase in O3. This method is still myopic in that when solving time period t , it does not look ahead to future time periods. We extend this work further to include a look-ahead feature to minimize secondary costs in future time periods.

The GRASP algorithm also includes neighbourhoods based on changing the firm of a single student, the order the specialties are studied by a student and swapping the allocation of two students in all or a subset of time periods. Feedback cannot be included in the traditional manner due to the problem of symmetry described previously. Therefore feedback is incorporated by keeping a large part of the solution from one cycle to another and only reconstructing the remaining parts of the solutions.

Experiments are conducted on both real and simulated datasets. They will demonstrate that the construction heuristic has a strong influence on the quality of the final solution, and that partial reconstructions are extremely effective. The constructions that incorporate an element of look ahead also proved to produce better results. The results generated are of high quality and significantly better than those produced manually.

References

1. Franz, L. S. and Miller, J. L., Scheduling medical residents to rotations: solving the large-scale multi-period staff assignment problem. *Opns. Res.*, 41, 269-279 (1993)
2. Pentico, D.W., Computer scheduling of medical school clerkships, *Computers and Education* 4, 139-143 (1980)
3. Pentico, D. W., Assignment problems: a golden anniversary survey. *Eur J. Opl. Res.*, 176, 774-793 (2007)
4. Fuller, E. J., Tackling scheduling problems using integer programming. MPhil Thesis, Swansea University (1998)

Scheduling Impatient Jobs with Diminishing Rewards

The Impact of Deteriorating Survival Probabilities on Mass Casualty Incident Triage

William L. Herring • Jeffrey W. Herrmann

1 Introduction

The development of mass casualty incident (MCI) response plans and protocols is an important priority for public health and hospital administrators, and managing the medical response to MCIs has received an increasing amount of attention in the trauma and emergency medicine literature. A review of the guidelines emerging from this literature, which rest primarily on retrospective statistical analyses and physicians' personal experiences, reveals a lack of consensus on many key points, including the precise objective of the response, the modification of treatment standards during the response, and, most prominently, the design and implementation of the triage process [7].

In a MCI, a large, potentially overwhelming, number of casualties are generated essentially simultaneously [3], [9]. In the most general sense, the objective of the medical response to a MCI is to do the "greatest good for the greatest number," and the triage process contributes to this goal by prioritizing a subset of the casualties for immediate treatment [4], [6]. Nearly all triage algorithms used in practice categorize patients based on their *need* for immediate treatment, reflecting the fact that patients may die while waiting for treatment [13]. The volume of casualties generated by a MCI forces triage algorithms to also consider whether patients will *benefit* from immediate treatment, since not all patients receiving treatment will ultimately survive [4]. Exactly how the likelihood of survival after treatment should be incorporated into triage decisions remains a subject of much debate in the MCI literature.

Mathematical modeling, and in particular discrete-event simulation, has emerged as a useful tool for the analysis of various aspects of response planning [2], [8], [10], [11], but relatively few studies have explicitly modeled the triage process. Sacco et al. use a linear programming formulation to optimize on-scene triage in the face of limited on-scene and transportation resources [15]. Argon et al., on the other hand, model the triage process as a single-server clearing system with impatient jobs (patients) that will abandon the system (die) if forced to wait too long for service (treatment) [1]. That work is one in a series of recent papers on

William L. Herring
Applied Mathematics & Statistics and Scientific Computation Program, University of
Maryland, College Park
E-mail: wherring@math.umd.edu

Jeffrey W. Herrmann
Department of Mechanical Engineering, Institute for Systems Research, University of
Maryland, College Park
E-mail: jwh2@umd.edu

scheduling impatient jobs in a clearing system [1], [5], [14]. The broader literature on stochastic job scheduling in queuing and clearing systems with abandonment extends well beyond these three articles, but they represent, to the best of our knowledge, the only research that considers the “MCI triage” clearing system. Both Argon et al. [1] and Li and Glazebrook [14] maximize the number of jobs that receive service using stochastic dynamic programming-based, state-dependent heuristics that outperform traditional static heuristics, suggesting that patient triage in the aftermath of a MCI should rely on state-dependent policies that take into account factors such as the number of casualties and resource availability.

In these papers all of the jobs receiving service make the same contribution to the overall system objective. In the context of MCI triage, even patients who receive treatment may not survive, and there is consensus in the literature that triage decisions should incorporate the impact of treatment on long-term patient survival [4], [13]. Furthermore, the widely accepted trauma concept of the “golden hour” suggests that patients’ long-term survival probabilities deteriorate as they wait for treatment [12], [15]. In order to investigate the role of deteriorating survival probabilities in MCI triage decisions, we incorporate them into the impatient job scheduling problem as diminishing rewards for service. The resulting objective, in the triage context, is to maximize the expected number of survivors (the sum of the survival probabilities of those patients who are treated), rather than the number of patients treated.

2 Impatient Jobs with Diminishing Rewards

The problem of scheduling impatient jobs in a single-server clearing system with diminishing rewards can be formally stated as follows. A single server is faced at time zero with a finite collection of jobs seeking service. Each job is associated with a random service time and a random lifetime, with the latter representing how long the job will wait for service before abandoning the system. The server receives a reward for each job that is serviced, but the magnitude of this reward may decrease as a function of the time that service is initiated. The server’s objective is to sequence the collection of jobs to maximize the expected reward earned, stopping only when all jobs have either been served or abandoned the system.

We assume that the jobs are divided into K classes, where class j is identified by a unique combination of service time (X_j) and lifetime (Y_j) random variables and a non-increasing reward function ($R_j(t)$). Let N_j be the number of jobs of class j present at time 0. We assume that the service and lifetime random variables, respectively, are i.i.d. within classes and independent between classes. The state of the system at each decision epoch t (at time 0 and at the conclusion of a job’s service) can be described by the number of jobs $\mathbf{n} = (n_1, n_2, \dots, n_K)$ of each class remaining in the system. We define the value function $V(\mathbf{n}, t)$ as the maximum expected reward earned for being in state $\mathbf{n}$ at decision epoch t . This leads to a stochastic dynamic programming (SDP) formulation with the following optimality equation and boundary condition:

$$\begin{aligned} V(\mathbf{n}, t) &= \max_{j|n_j > 0} \{R_j(t) + E[V(\mathbf{n}', t')]\} \\ V(\mathbf{0}, t) &= 0 \end{aligned} \quad (1)$$

In this formulation, t' is the decision epoch at the completion of service for a job of class j , $\mathbf{n}'$ is the number of jobs remaining in the system at time t' (accounting for abandonments during service), and the expected value is taken with respect to class j ’s service time and the lifetimes for all other jobs.

In the general case (general service time and lifetime distributions and no additional assumptions about the reward functions), the need to explicitly include all potential decision epochs (which can be any $t \geq 0$) in the state space ensures that this SDP will be

computationally intractable, even for a reasonable discretization of the time dimension. Even in the Markov case (exponential random variables), strong assumptions are required on the reward functions in order to make the computational effort manageable. These limitations motivate the search for heuristic scheduling rules, which results in the extension of the heuristics found in the existing impatient jobs literature and the development of new heuristics.

3 Scheduling Heuristics and Computational Testing

Our analysis develops a range of heuristics for generating good solutions to the problem stated above. Because the problem is dynamic, the heuristics are similar to dispatching rules used in other scheduling disciplines that identify the class from which a job should be selected at each decision epoch (when the server is available). In the following discussion, let the service times and lifetimes be exponentially distributed with means μ_j^{-1} and r_j^{-1} , respectively, $\lambda_j(t)$ be the hazard rate associated with the reward function $R_j(t)$, and δ_{ij} be the Kronecker delta.

The shortest expected processing time (SEPT) heuristic prioritizes the jobs that are easiest to complete. Our dynamic $Rr\mu$ rule (motivated by Glazebrook et al. [5]) gives priority to the class with the largest value of $R_j(t)r_j\mu_j$ at each decision epoch. We also extend this rule to consider the rate at which rewards are diminishing; the $R(r + \lambda)\mu$ rule gives priority at each decision epoch to the class with the largest value of $R_j(t)(r_j + \lambda_j(t))\mu_j$. Argon et al. [1] propose the TRI rule that minimizes the approximate expected number of jobs that will abandon the system during the service of the next job. We modify this approach to create the RTRI rule, which gives priority to the class with the smallest value of the following expression:

$$\frac{1}{R_j(t)} \left(1 + \frac{1}{\mu_j + \lambda_j(t)} \sum_{i|n_i - \delta_{ij} > 0} (n_i - \delta_{ij}) r_i \right) \quad (2)$$

The Reward-based Minimum Losses During Service (RMLDS) rule gives priority to the class with the smallest value of the following expression, which equates to the amount of reward lost during the service of the next job:

$$R_j(t) + \sum_{i|n_i - \delta_{ij} > 0} (n_i - \delta_{ij}) \left[R_i(t) - \int_0^\infty \mu_j e^{-(r_i + \mu_j)s} R_i(t + s) ds \right] \quad (3)$$

We also studied a fluid-based approximate dynamic programming approach (similar to that presented in [14]) but will not present that approach here.

To test our heuristics we generate a realistic set of problems. Because the most important distinction during a MCI is between those who require immediate care and those who do not [4], our analysis focuses on problems with just two job classes. For the mean service times and long-term survival probabilities, we use values provided by the MCI literature [10], [11], [15]. Because the literature contains very little data on how long patients can survive while waiting for treatment, we considered a wide range of what we consider to be reasonable lifetimes.

We first create 3,456 problem instances with rewards that decay exponentially to zero at a uniform rate ($\lambda_1(t) = \lambda_2(t) = \lambda$), and, for each problem instance, the performance of each rule is averaged over 5,000 simulation replications. This initial set of problems can be solved optimally, so the heuristics are compared with the optimal solution and with each other. We then create 4,320 problem instances with more general exponentially decaying rewards (which cannot be solved optimally), and, for each problem instance, the performance of each rule is averaged over 2,500 simulation replications.

The SEPT, RTRI, and RMLDS rules significantly outperform the $Rr\mu$ and $R(r + \lambda)\mu$ rules. For problems with constant rewards ($\lambda = 0$), the average performance of the SEPT, RTRI, and RMLDS rules deviates from the optimal performance by 1.5%. The average performance of the $Rr\mu$ rule and the $R(r + \lambda)\mu$ rule is 2.6% from the optimal.

For problems with uniformly decaying rewards, the average performance of the SEPT, RTRI, and RMLDS rules deviates from the optimal by less than 0.1% for both $\lambda^{-1} = 180$ and $\lambda^{-1} = 60$. The average performance of the $Rr\mu$ rule is 23% from the optimal for $\lambda^{-1} = 180$ (37% for $\lambda^{-1} = 60$), and the average performance of the $R(r + \lambda)\mu$ rule is 3.8% from the optimal for $\lambda^{-1} = 180$ (0.7% for $\lambda^{-1} = 60$).

For problems with more general exponentially decaying rewards, the average performance of the SEPT, RTRI, and RMLDS rules deviates from the best performance observed by less than 1%. The average performance of the $Rr\mu$ rule is 9% from the best observed, and the average performance of the $R(r + \lambda)\mu$ rule is 4% from the best observed.

In conclusion, we find that in many cases it is preferable to prioritize less-critical patients (with shorter expected treatment times and higher long-term survival probabilities) in spite of the fact that their survival probabilities deteriorate more slowly and they can survive much longer while waiting for treatment. In cases where the survival probabilities are more equal and the less-critical patients are capable of surviving for quite some time without treatment, we find that it becomes preferable to prioritize the more critically wounded patients.

Acknowledgements The authors were supported by the Montgomery County, Maryland Advanced Practice Center at the Montgomery County, Maryland, Department of Health and Human Resources and an OR of the Future grant from the US Army Medical Research Acquisition Activity.

References

1. Argon, N.T., Ziya, S., Righter, R. Scheduling impatient jobs in a clearing system with insights on patient triage in mass casualty incidents, *Probability in the Engineering and Informational Sciences*, 22, 301-332 (2008)
2. Christie, P.M.J., Levary, R.R. The use of simulation in planning the transportation of patients to hospitals following a disaster, *Journal of Medical Systems*, 22(5), 289-299 (1998)
3. Frykberg, E.R. Medical management of disasters and mass casualties from terrorist bombings: how can we cope? *Journal of Trauma*, 53(2), 201-212 (2002)
4. Frykberg, E.R., Triage: principles and practice, *Scandinavian Journal of Surgery*, 94, 272-278 (2005)
5. Glazebrook, K.D., Ansell, P.S., Dunn, R.T., Lumley, R.R. On the optimal allocation of service to impatient tasks, *Journal of Applied Probability*, 41, 51-72 (2004)
6. Halpern, P., Tsai, M.-C., Arnold, J.L., Stok, E., Ersoy, G. Mass-casualty, terrorist bombings: implications for emergency department and hospital emergency response (part II), *Prehospital and Disaster Medicine*, 18(3), 235-241 (2003)
7. Herring, W.L. Medical response planning and triage for mass casualty terrorist bombings, Technical Report 2010-18, Institute for Systems Research, University of Maryland, College Park, MD (2010)
8. Hirshberg, A., Stein, M., Walden, R. Surgical resource utilization in urban terrorist bombing: a computer simulation, *Journal of Trauma*, 47(3), 545-550 (1999)
9. Hirshberg, A., Holcomb, J.B., Mattox, K.L. Hospital trauma care in multiple-casualty incidents: a critical review, *Annals of Emergency Medicine*, 37(6), 647-652 (2001)

10. Hirshberg, A., Scott, B.G., Granchi, T., Wall, M.J., Mattox, K.L., Stein, M. How does casualty load affect trauma care in urban bombing incidents? A quantitative analysis, *Journal of Trauma*, 58(4), 686-695 (2005)
11. Hupert, N., Hollingsworth, E., Xiong, W. Is overtriage associated with increased mortality? Insights from a simulation model of mass casualty trauma care, *Disaster Medicine and Public Health Preparedness*, 1(1 Suppl), S14-S24 (2007)
12. Lerner, E.B., Moscati, R.M. The golden hour: scientific fact or medical “urban legend”? *Academic Emergency Medicine*, 8(7), 758-760 (2001)
13. Lerner, E.B., Schwartz, R.B., Coule, P.L., Weinstein, E.S., Cone, D.C, Hunt, R.C., Sasser, S.M., Liu, J.M., Nudell, N.G., Wedmore, I.S., Hammond, J., Bulger, E.M., Salomone, J.P., Sanddal, T.L., Lord, G.C., Markenson, D., O’Connor, R.E. Mass casualty triage: an evaluation of the data and development of a proposed national guideline, *Disaster Medicine and Public Health Preparedness*, 2(Suppl 1), S25-S34 (2008)
14. Li, D., Glazebrook, K.D. An approximate dynamic programming approach to the development of heuristics for the scheduling of impatient jobs in a clearing system, *Naval Research Logistics*, 57, 225-236 (2010)
15. Sacco, W.J., Navin, D.M., Fiedler, K.E., Waddell, R.K., Long, W.B., Buckman, R.F. Precise formulation and evidence-based application of resource-constrained triage, *Academic Emergency Medicine*, 12, 759-770 (2005)

Operational Surgery scheduling problems

Modelling and solving

Atle Riise • Carlo Mannino

1 Introduction

The term surgery scheduling is used about a variety of strategic, tactical and operational scheduling problems [1], many of which are critical to an efficient use of hospital resources. Our focus is on operational surgery scheduling, which may be informally described as the task of assigning times of surgery for each patient, while reserving capacity for these surgeries on a set of constrained renewable resources. Such resources may be, for example, operating rooms, operation teams, surgeons, equipment, or post-operative bed capacity. Objectives are typically tardiness costs (overtime), hospitalization costs, intervention costs, operating room utilization, patient's waiting time, and patient or personnel preferences, among others. These scheduling problems are often NP-hard (see e.g. [2]). The exact problem formulation varies substantially between hospitals, or even hospital departments. In addition, the degree of detail vary between different planning situations; patient admission planning may consider only one or two kinds of resources, is mainly concerned with allocating a date of admission for each patient, and typically has a long time horizon. Closer to the day of surgery, such as when scheduling surgeries for the next day or week, the number of activities, resources and choices to make increase. This diversity presents a challenge for those who wish to create scheduling methods that are generally applicable to surgery scheduling problems. We approach this challenge by creating a new, general model for surgery scheduling problems. The model can be seen as a rich extension to the resource constrained project scheduling problem (RCPSP) [3]. This generalised reference model will enable research into solution methods that are robust across problem variations. We demonstrate how the surgery scheduling problem is naturally modelled as an extension to a classical disjunctive graph model. In order to investigate the limitations of complete methods for solving such problems, we also show how this model can be formalized as a mixed integer model. Using realistic test data for three typical planning situations, we conduct benchmarking experiments using a commercial mathematical programming tool. We conclude with a discussion of the limitations of this approach and of promising directions for future research.

Atle Riise
SINTEF ICT/University of Oslo
E-mail: atle.riise@sintef.no

Carlo Mannino
SINTEF ICT/University of Oslo
E-mail: carlo.mannina@sintef.no

2 Modelling the surgery scheduling problem

2.1 A generalised disjunctive graph model

The many variations of the surgery problem are reflected in the literature; no two papers address the same problem, and, to our knowledge, little effort has been made to create a generalised reference model. We show that the generalised surgery scheduling problem can be naturally modelled as an extension of the classical RCPSP, with multiple projects, multiple resources per activity, multiple modes, time dependent resource availability and capacity, block constraints, setup times, mode dependent precedence constraints, mode restriction, project disjunctions, and multi-capacity resources. The problem is naturally modelled as a directed activity-on-node graph, where arc lengths represent activity durations (and possibly added resource setup times), and each resource arc have capacity and flow or resource units. The basic concepts of our model are:

- *Project*. The treatment of each patient is modelled as a project, consisting of a set of activities, some of which are connected by precedence constraints.
- *Activity*, which is the various actions that have to be planned for each patient, such as preparation of the patient, surgery, cleaning the operating room, etc.
- *Resource*, which is a generalisation of all relevant resources, such as operating rooms, teams, surgeons, recovery units, etc.
- *Resource Period*. This is a concept without any direct equivalent in the traditional RCPSP formulations. A sequence of resource periods is defined for each resource to model time dependent resource availability and capacity in a flexible manner.
- *Mode*. For each activity, a set of feasible modes are defined out of which one has to be selected in a feasible solution. A mode defines a possible choice of resources that can perform the activity, the demand for each resource, and the mode-dependent activity duration. In addition, we extend this common concept of a mode to include mode-dependent precedence constraints.

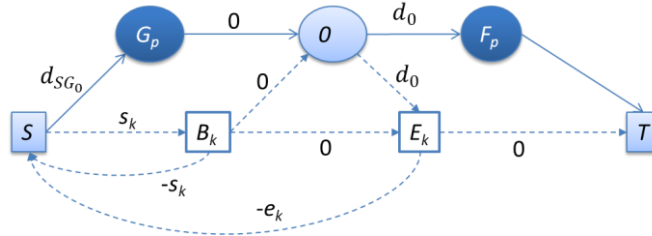
In classical disjunctive graph models, disjunctions are simply pairs of directed arcs in the problem graph, out of which exactly one must be selected in a feasible solution. We introduce a generalisation of the concepts by defining a “disjunctive set” as a set of sets of arcs, out of which exactly one set of arcs must be chosen. Our scheduling problem can then be loosely formulated as follows:

Given a set of resources, a set of available periods for each resource, a set of projects, a set of activities for each project, and a set of modes for each activity, find:

- An allocation of modes to activities
 - A schedule for each resource, that:
 - Includes all activities whose mode uses the resource
 - Respects a set of precedence constraints between project activities
 - Respects a set of disjunctive constraints, represented by a collection of disjunct sets
- , so that some cost function is minimized.

One may regard the problem as decomposed into a routing and a scheduling problem. The routing problem is to select the resources (i.e. the mode) to use for each activity. Evaluating the feasibility and the objective value associated with such an assignment involves solving the scheduling problem. The scheduling problem can be modelled as a directed graph with one node for each project activity. The arcs of the graph are the union of sub graphs, one

for each project and one for each resource. Project sub graphs represent the precedence constraints among project activities and the length of the arcs are set to the mode-dependent duration of the source activity of each arc. Resource graphs contain a sequence of resource periods, each with a specified start and end time represented by artificial nodes. A range of constraints and objectives can be calculated directly from the propagated start time of the end nodes of these periods. The figure shows a simple example with one project graph with a single activity (“0”) that is inserted into one resource graph in (the only) resource period k . The nodes G_p and F_p are the start and end nodes of the project graph, while the nodes B_k and E_k are the start and end nodes of the resource period k . The arcs lengths represent activity durations or time constraints.



The choice of resource period for an activity is represented as a disjunctive set in the problem graph. Normal disjunctive constraints between activities using the same resource are also modelled as disjunctive sets. Arc lengths in the resource graph are based on activity durations, with the possible addition of resource setup times. Resource capacity is modelled as an initial flow of resource units into each resource period, and can thus vary from period to period. This is useful in a long term planning setting, where capacity must be reserved for scheduling patients that are not yet known. A feasible solution is in principle created by choosing arc sets from each disjunctive set, resulting in a solution graph without disjunctions. In this process, flow must be maintained on the resource graph arcs in a way that ensures flow conservation and meets resource demands.

3 Solving the surgery scheduling problem

Perhaps reflecting the variety of real world surgery scheduling problems, the literature includes the use of a wide range of resolution methods, both exact and heuristic [4]. Many variants of the problem are NP-hard, and several authors conclude that a meta-heuristic approach is needed for problems of realistic size and complexity [5]. As we present a new generalised model, however, we wish to investigate what the limitations are for exact methods when solving this model for typical and realistic problem instances. Long term planning tasks typically contain fewer activities per project, consider fewer resources, and can use a somewhat simplified set of constraints from the reference model. Very short term surgery scheduling, on the other hand, will be very detailed in activities and resource use, and may consider all the constraints in the reference model. As a basis for our investigation, we constructed three sets of benchmark problems, each from a different planning situation (“admission planning”, “weekly surgery scheduling”, and “daily surgery scheduling”). In principle the problem is multi-objective. In our experiments, however, we minimize a weighted sum of the individual objective components. All benchmark instances are based on real hospital data.

3.1 A Mixed Integer Program

We present a mixed integer program for the entire problem, including both routing and sequencing variables. The rich nature of this real world problem makes this a quite extensive model in terms of types of constraints and the number of input parameters. Time is treated as a continuous variable, while all other variables are binary.

We use the mathematical programming tool CPLEX for our experiments. Based on the experimental results, we discuss the practical limitations of mathematical programming, as it is

applied to our model, in terms of problem complexity and size. Our conclusion is that for a number of realistic problems, solving by mixed integer programming is not practically feasible, and that heuristic approaches should be investigated if one wishes to produce a generally applicable solver for the generalized surgery scheduling problem.

Acknowledgements This work is supported by the Research Council of Norway, through the HOSPITAL project. The authors wish to thank Prof. Edmund K. Burke and Dr. Geir Hasle for valuable discussions and suggestions.

References

1. Blake, J.T. and M.W. Carter, *A goal programming approach to strategic resource allocation in acute care hospitals*. European Journal of Operational Research, 2002. **140**(3): p. 541-561.
2. Hans, E., et al., *Robust surgery loading*. European Journal of Operational Research, 2008. **185**(3): p. 1038-1050.
3. Hartmann, S. and D. Briskorn, *A survey of variants and extensions of the resource-constrained project scheduling problem*. European Journal of Operational Research, 2010. **207**(1): p. 1-14.
4. Cardoen, B., E. Demeulemeester, and J. Beliën, *Operating room planning and scheduling: A literature review*. European Journal of Operational Research, 2010. **201**(3): p. 921-932.
5. Riise, A. and E. Burke, *Local search for the surgery admission planning problem*. Journal of Heuristics, 2010: p. 1-26.

Decomposition approaches for multi-stage multi-product batch-production scheduling: a refined batching model

Norbert Trautmann · Philipp Baumann ·
Nadine Saner · Tobias Schäfer

1 Planning problem

In batch production systems, value is added to material by successive transformation tasks such as chemical reactions, unification, and separation. We are concerned with the case of multi-product systems, which are usually chosen in make-to-order production when the product range spans various product families with relatively small demands. Each product family requiring a specific plant configuration, short customer lead times and high resource utilization can only be attained by minimizing the production makespan.

Batch production means that the total requirements for the final products and the intermediates are split into batches. To process a batch, first the inputs are loaded into a processing unit, then a transformation process (called task in the following) is executed, and finally the output is unloaded from the unit. We consider multi-purpose processing units, which can operate different tasks, but only one at a time. Between consecutive tasks operated by the same processing unit and before idle times of a processing unit, a cleaning of the unit with sequence-dependent duration may be necessary. The minimum and maximum filling levels of a processing unit give rise to a lower and an upper bound on the batch size. The duration of a task depends on the processing unit used, but not on the batch size. In general, the intermediates can be stored in storage facilities of finite capacity; some intermediates, however, are perishable and must be consumed immediately after production. The planning problem studied in this paper consists in computing a feasible schedule such that a given demand is fulfilled and the production makespan is minimized.

2 Contribution

Due to the computational intractability of the problem described in Section 1, various decomposition approaches have been proposed in the literature. For example, [3]

Norbert Trautmann · Philipp Baumann · Nadine Saner · Tobias Schäfer at University of Bern,
Department of Business Administration, 3012 Bern, Switzerland
E-mail: norbert.trautmann@pqn.unibe.ch

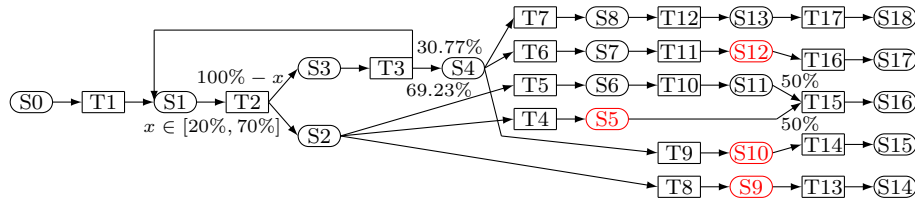


Fig. 1 State-task network of the Westenberger-Kallrath sample process

decompose the planning problem into a batching subproblem, which consists in computing a set of batches with fixed sizes, and a batch-scheduling subproblem, which consists in allocating the production equipment to the processing of these batches over time. Such decomposition strongly decreases the CPU time required for computing a good feasible solution.

The batching model proposed by [3] implies equal batch size (strategy E) and equal input/output proportions for all executions of a task. In the present paper, we show by means of an experimental study that therefore, even when both subproblems are solved to optimality, the makespan obtained may be longer than in an optimal solution obtained by an integrative approach. Moreover, we show how this gap can be reduced by computing appropriately an individual batch size (strategy I) for each execution of a task. In detail, we compare the following alternative solution approaches:

- Integrative approach (approach I): we solve the problem by using the integrative formulation of [1]. The schedule obtained represents an optimal solution to the instance at hand.
- Decomposition approach with batching strategy E (approach DE): we solve the batching subproblem as proposed in [3] as a mixed-binary linear program, and we solve the corresponding batch-scheduling subproblem using the formulation of [1].
- Decomposition approach with batching strategy I (approach DI): first, we solve the batching subproblem as proposed in [3] as a mixed-binary linear program. The solution obtained defines the number of executions, the batch size, and the input/output proportions of each task. Then, we solve the LP presented in this talk for adapting the individual batch sizes for each execution of each task. Last, we solve the resulting batch-scheduling subproblem using the formulation of [1].

3 Results

For our computational analysis, we used the Westenberger-Kallrath sample production process [2]. Fig. 1 shows the state-task-network of the process; detailed information about the process and the test instances is provided in [1]. Our computational results for these instances indicate that approach DI allows to compute schedules with shorter makespans than approach DE.

Figures 2–4 exemplify the schedules obtained for problem instance BG02. Here, applying the LP model proposed in this talk reduces the gap between the optimal solution (approach I) and the solution obtained by approach DE by 1/3.

Acknowledgements This work was supported by the Swiss National Science Foundation, grant 205121-125106.

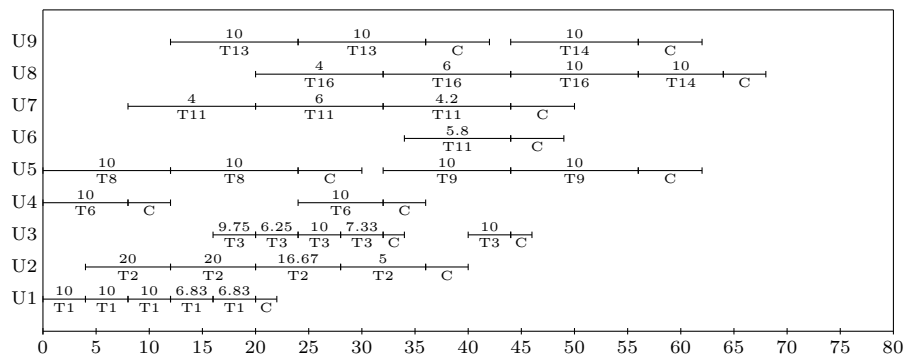


Fig. 2 Schedule obtained with approach I (makespan: 68)

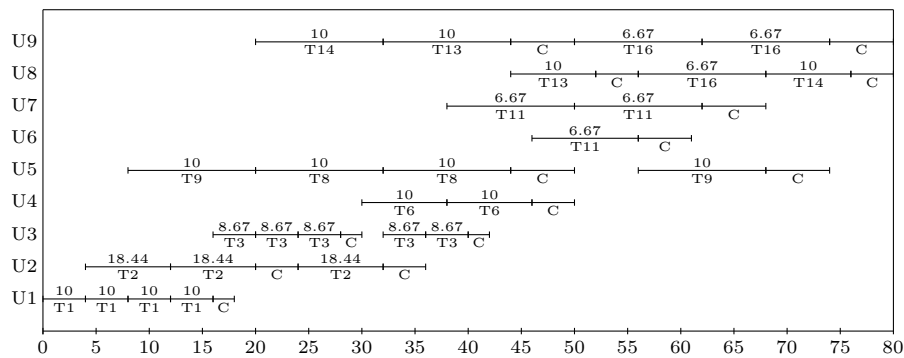


Fig. 3 Schedule obtained with approach DE (makespan: 80)

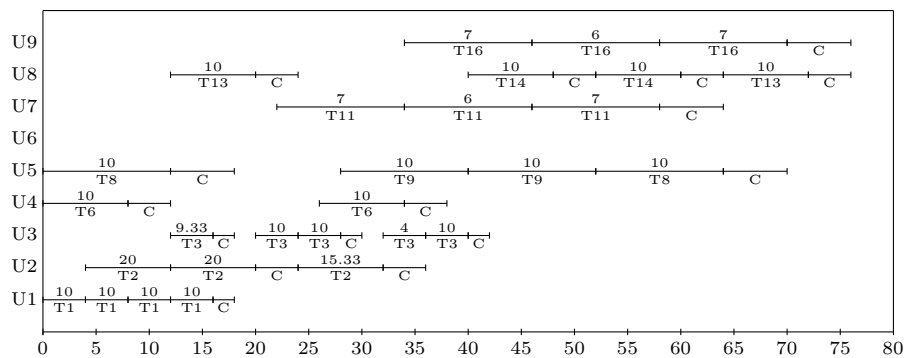


Fig. 4 Schedule obtained with approach DI (makespan: 76)

References

1. Blömer, F., Günther, H.: Scheduling of a multi-product batch process in the chemical industry. *Computers in Industry* **36**, 245–259 (1998)
2. Kallrath, J.: Planning and scheduling in the process industry. *OR Spectrum* **24**, 219–250 (2002)
3. Neumann, K., Schwindt, C., Trautmann, N.: Advanced production scheduling for batch plants in process industries. *OR Spectrum* **24**, 251–279 (2002)

Flexible Constraint Optimization of Product Profits

A responsive approach to determine profit-optimal product start rates subject to mean cycle time and product volume constraints in support of production planning for semiconductor fabrication facilities

**Detlef Pabst ·
John W. Fowler ·
Gerald T. Mackulak.**

1 Introduction

Production planning of semiconductor fabrication facilities (fabs) in the foundry business engages an "Available to Promise" process for assessing the fab's ability to accept new orders and a "Real Time Order Management" process for assessing the fab's ability to conform to changes on conditions of already accepted orders. These business processes are subject to challenging requirements: Respond in a fast time, ensure of the fab's continued ability to satisfy prior commitments made for other existing orders, and leverage the customer-negotiable leeway in the order conditions to optimize the fabs profit margin. In support of such tasks this paper proposes an approach that time-efficiently optimizes a fab profit measure over a limited product start rate domain under varying what-if scenarios that are specified as constraints for mean product cycle times and product volume combinations, or differing product profit contributions.

Since the complexity of a multi-product semiconductor manufacturing process typically requires a time-consuming simulation analysis to obtain reliable mean product cycle time estimates, a classical approach in the described production planning situation is to ensure both the necessary short response time and the fabs ability to comply with the made commitments by combining conservative product cycle time estimates with static, fast capacity estimates. As this classical approach is limiting to the profit margin optimization opportunity, we are proposing an algorithm that extends the procedure published in [1] for modeling the mean of the steady state product cycle times in dependence of individual product start rates, i.e. as function of throughput and product mix. The model fitting procedure combines the prediction strength of simulation analysis with the scenario-agility of non-linear regression prediction in order to map

Detlef Pabst
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University E-mail: Detlef.Pabst@asu.edu

John W. Fowler
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University E-mail: John.Fowler@asu.edu

Gerald T. Mackulak
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University E-mail: Gerald.Mackulak@asu.edu

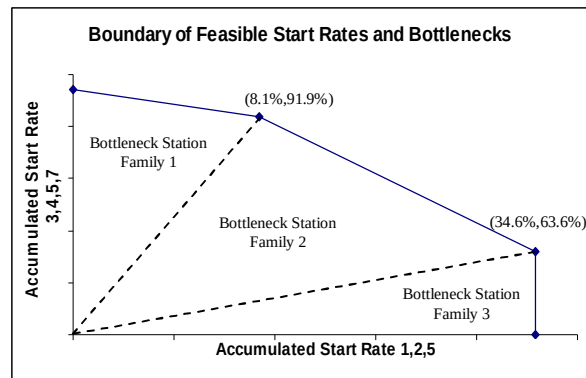


Fig. 1 Constant Bottleneck Regions in the Accumulated Start Rate Domain.

out the cycle time, throughput, and product-mix relation. Using such a prior-prepared response surface we design an optimization methodology that maximizes a profit measure, expressed as weighted sum of the product start rates, under consideration of constraints on the product start rates and on the means of the steady-state product cycle times. Having completed the simulation and model fitting effort in advance, our approach allows executing differing constraint-scenarios within the short time frame imperative in the described planning situation.

2 Basic Approach

The multi-stage estimation procedure of [1] for the mean product cycle time in dependence of throughput and product mix fits a series of queueing analysis motivated non-linear regression models to data generated by a sequence of designed simulation experiments. The resulting response surface provides an estimation error controlled prediction of the mean product cycle time for any given set of product start rates within the fitted domain.

Given a set of product start rate and cycle time constraints, the proposed method uses such a non-linear cycle time response surface to translate the cycle time constraints into a hierarchy of piece-wise linear approximations for the constant mean cycle time surfaces within the product start rate domain. These approximations are locally refined during the progress of a standard simplex basis exchange algorithm that maximizes a linear function in the product start rate variables.

3 Graphical Example for the Problem Definition

An application of the proposed procedure is illustrated for the case of a reference fab simulation model with 7 products falling into two technology groups: A full flow of products 1, 2, 3 and a short loop of products 3, 4, 6, 7. With changes in the ratio of the group start rates (product mix) the overall fab capacity becomes limited by the throughput capabilities of different station families (bottlenecks). Figure 1 shows the constant bottleneck regions for the considered simulation model.

We define the product profit weights for the two product groups approximately in the ratio 6 to 1, proportional to their manufacturing complexity as measured by the

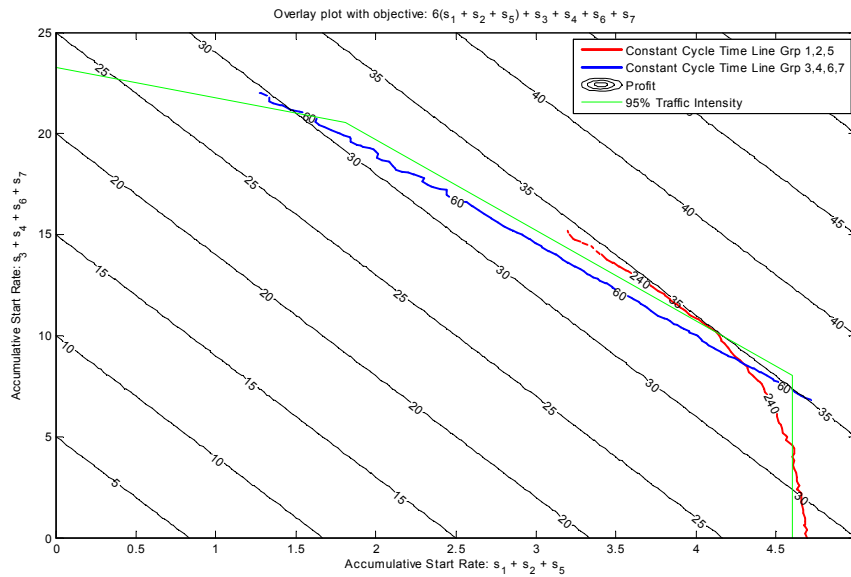


Fig. 2 Overlay Plot of Cycle Time Contours with a Profit Model Contours (Using Averaged Objective Coefficients and Accumulated Start Rates).

route lengths or the total processing times. The objective is to maximize the sum of the weighted product start rates such that constraints on the average product cycle times, product start rates and product percentages are satisfied as listed in Table 1, as well as a limit for the overall plan fab utilization of 95%.

Engaging the pre-generated cycle-time-throughput-product-mix model for the two product groups, our approach translates the constraints on the average product cycle time into non-linear constraints on the product start rates. Figure 2 sketches these constraints and overlays the resulting feasible start rate domain with a contour plot of the profit objective in order to indicate the optimal product group start rates. Table 1 provides the approximate optimal values for the individual product start rates. After the 2% bounds are satisfied for the low profit products 1, 3, 6, the remaining start rates of the high profit products 5 and 7 are maxed out until the cycle time constraints are binding.

References

1. F. Yang, J. Liu, B. L. Nelson, B. E. Ankenman and M. Tongarlak, Metamodeling for Cycle Time-Throughput-Product Mix Surfaces using Progressive Model Fitting, Production Planning and Control, forthcoming.

Product	1	2	5	3	4	6	7
Profit (\$ per Wf)	54000	60000	66000	9000	8000	11000	12000
Min. Percentage	2%			2%		2%	
Upper CT Bound	240hr	240hr	240hr	60hr	60hr	60hr	60hr
Max. Start Rate			3 Wf/hr				
Optimal Solution	0.258	1.022	3	0.258	0	0.258	8.085

Table 1 Optimization Problem and Solution.

Two-Stage Shipment Planning with Uncertainties: A Stochastic Programming Approach

Yer Van Hui, Jia Gao, Lawrence C K Leung and Stein W. Wallace

1 Introduction

Airfreight forwarders are third-party operators responsible for handling a majority of international tonnage by using in-house resources, partners, and subcontracting agents. To ensure that client shipments could be delivered on time and at a low cost, forwarders commonly design a shipment plan from a holistic perspective. A shipment plan usually consists of a number of shipments (jobs), and each job has a number of processing activities such as trucking, warehousing and air transporting. If several consecutive activities of a job are assigned to the same agent, we call it integration. If the same activities from several shipments are processed together by a single agent, it is called consolidation. Effective integration and consolidation are crucial to achieving cost- and time-efficiency. Integration achieves setup cost and setup time reduction, but it might forsake the use of a more efficient processing agent elsewhere (Wong et al. [6]). Consolidation is the primary means used to reduce shipping costs by achieving better utilization of resources (Bookbinder and Higginson [1] and Çetinkaya and Lee [2]). However, it might lead to increased inventory costs, delays and longer routes.

International airfreight shipments are normally multi-modal in the form of ground-air-ground transport. Throughout the shipping process, there exist many uncertain elements. Delaying factors such as bad weather, customs, carrier overbooking, system malfunctioning, and traffic condition might have significant effects on the reliability of a shipment. Both integrations and consolidations affect a shipment's reliability as well. Especially for a

Yer Van Hui
City University of Hong Kong
E-mail: msyervan@cityu.edu.hk

Jia Gao
City University of Hong Kong
E-mail: gjtina@cityu.edu.hk

Lawrence C K Leung
Chinese University of Hong Kong
E-mail: lawrence@baf.msmail.cuhk.edu.hk

Stein W. Wallace
Lancaster University
E-mail: stein.w.wallace@lancaster.ac.uk

shipment requiring consolidations, its reliability depends not on its own processing randomness but also on the processing conditions of other jobs it consolidated with.

In practice, an increasing number of international forwarders have implemented different types of real-time monitoring techniques to better control the shipping process with uncertainty. When a shipment is expecting a serious delay that would miss the target delivery date, a forwarder might make adjustments to the current shipment plan for the subsequent shipping activities. Past shipment planning models were deterministic optimization models (Leung et al. [5] and Wong et al. [6]). One major shortcoming of such a model is that it fails to incorporate in-process adjustments of a shipment plan. We propose to address the design of a shipment plan under uncertainty using a stochastic programming approach.

2 A Two-Stage Stochastic Shipment Planning Model

It is assumed that each job contains a number of activities selected from a set of activities. Job activities are divided into two groups. The first J_1 activities $A_1, A_2, \dots, A_{J_1}$ are conducted in stage one and the remaining J_2 activities belong to stage two. Assume that the processing time T_{ijk} of activity j of job k assigned to agent i is random with a probability distribution function $f(x_{ijk})$ for activities in stage one. For illustrative purpose, the processing times for activities in stage two are assumed fixed.

A two-stage stochastic shipment planning model is considered to obtain the optimal shipment strategy for a forwarder, which is built on the premise that the forwarder has the in-process option to adjust the shipment plan in stage two based on the realization of activities in stage one. When a shipment is experiencing serious delay that threatens missing the target delivery date, the forwarder might make adjustments on the assigned agents and routings for activities in stage two.

Under the two-stage scheme, we could construct the objective cost function for the two-stage model given as

$$\begin{aligned} & \text{Min}_{x_{ijk} \text{ in stage 1}} \{ \text{Expected total processing cost} \} \\ & = \text{Min}_{x_{ijk} \text{ in stage 1}} \left\{ \int_{x_{ijk} \text{ in stage 1}} [\text{Total processing cost} | x_{ijk}] f(x_{ijk}) dx_{ijk} \right\}. \end{aligned} \quad (1)$$

For each of the selected set of assignments $\{x_{ijk}\}$ and a realization in stage one, the total processing cost in completing the assigned jobs is

$$\text{Min}_{x_{ijk} \text{ in stage 2}} \left\{ \sum_i \sum_j \sum_k (s_{ijk} + a_{ijk}) x_{ijk} - \sum_m w_m y_m - \sum_n r_n z_n \right\}, \quad (2)$$

subject to the updated constraint set at the beginning of stage two. Here s_{ijk} and a_{ijk} are the setup cost and processing cost of activity j of job k if assigned to agent i respectively; y_m includes the set of job integrations implemented in stage one and the updated job integrations in stage two with the corresponding setup cost savings w_m ; and z_n embraces the set of job consolidations implemented in stage one and the updated job consolidations in stage two with the corresponding cost savings r_n . Note that in this two-stage model, the shipment planning decisions are for the job activities in stage one. The decision on the assignment of activities in stage two is subject to the assignments and realizations in stage one as well as the available resources (agent assignments, routings, job integration sets, consolidation sets, and capacities) at the beginning of the second stage.

3 Scenario Generation and Optimization Process

We suggest a scenario tree approach for the solution of the two-stage stochastic shipment planning problem. Detailed discussion of this approach and its stability requirements are given in Kaut and Wallace [4]. A scenario tree $\{X_\ell\}$, $\ell = 1, \dots, p$ represents different states of the shipment plan after the assigned activities in stage one are performed with processing times $\{t_{ijk}^\ell\}$, $\ell = 1, \dots, p$. With the scenario tree representation in modeling the uncertainties in stage one, the problem remains a mixed 0-1 linear programming model where the objective function is the weighted average processing cost in all scenarios. The following two-level optimization is proposed.

Sub-problem:

- Step 1. Select a set of agent assignments $\{x_{ijk}\}$ for job activities in stage one.
- Step 2. Construct a scenario tree $\{X^{(\ell)}\}$, $\ell = 1, \dots, p$ for the processing time distributions in stage one.
- Step 3. For a given scenario $\{X^{(\ell)}\}$, find the best available solution to complete the activities in stage two and record the optimal cost $F^{(\ell)}$.
- Step 4. Repeat step 3 for all scenarios and calculate the weighted average cost F given $\{x_{ijk}\}$.

Global optimization:

Find the optimal agent assignments in stage one to minimize the objective cost $F(\{x_{ijk}\})$.

Note that the scenario tree approach can deal with inconsistencies in the specifications of processing time distributions (Høyland and Wallace [3]). The proposed model also enables decision makers to specify worst-case events to be included in the scenarios.

When the processing times of shipment activities are deterministic, a Tabu-based solution can be found in Wong et al. [6]. We develop a Tabu-based algorithm adopting the unique characteristics of shipment planning to search for the optimal solutions in the two-stage problem. Details covering the generation of the initial shipment plan, moving methods and managerial considerations will be discussed in an illustrative example.

References

1. Bookbinder, J. H., Higginson, J. K., Probabilistic modeling of freight consolidation by private carriage, *Transportation Research Part E: Logistics and Transportation Review*, 38, 305-318 (2002).
2. Çetinkaya, S., Lee, C. Y., Optimal outbound dispatch policies: Modeling inventory and cargo capacity, *Naval Research Logistics*, 49, 531-556 (2002).
3. Høyland K and Wallace, S. W., Generating scenario trees for multistage decision problems, *Management Science*, 47(2), 395-307 (2001).
4. Kaut, M and Wallace, S. W., Evaluation of scenario-generation methods for stochastic programming, *Pacific Journal of Optimization* 3(2), 257-271 (2007).
5. Leung, L. C., Hui, Y. V., Wang, Y., Chen, G., A 0-1 model for the integration and consolidation of air cargo shipments, *Operations Research*, 57, 402-412 (2009).
6. Wong, W. H., Leung, L. C., Hui, Y. V., Airfreight forwarder shipment planning: A mixed 0-1 model and managerial issues in the integration and consolidation of shipments, *European Journal of Operational Research*, 193, 86-97 (2009).

Facility layout with random demand

A reduction to the classical deterministic QAP

Stein W. Wallace • Yifei Zhao

Facility layout problems (FLPs) are studied to determine locations for facilities (machines) within a factory to achieve a minimum material handling cost (transport cost) [1]. In 1957, Koopmans and Beckman were the first to formulate FLPs as mathematical optimization models – resulting in the QAP (Quadratic Assignment Problem) [2]. The QAP is widely used but one of its assumptions is that flow between any two machines is known. The problem is NP-hard [3] and many efforts have been made to develop algorithms, both exact and heuristic, to solve the QAP. Exact methods are used to solve small-sized problems (less than 40 facilities) while heuristics are used for large-sized problems often seen in industrial applications. Simulated annealing (SA), genetic algorithm (GA) and tabu search are commonly used [4].

In 2000, a FLP was formulated as a layout-flow model by Benjaafar and Sheikhzadeh [5]. This model is an extension of the QAP, with the main difference being that capacities of machines are taken into consideration. In such a layout-flow model, there are several machine types each of which has multiple copies. One of the assumptions is that flow between any two machine *types* is known but how to assign flow among copies of each type needs to be decided to achieve minimum transport cost. Because of this, the layout-flow model has two parts: a facility layout problem and a flow assignment problem.

In [5], Benjaafar and Sheikhzadeh also introduced stochastic demands into the layout-flow model making another significant step forward from the original formulation. This flow-layout model can be used to design layouts for factories which face uncertain requirements. Uncertainty comes from two aspects: random demand levels and uncertain sequences of machine types. The latter typically comes from changes in product specifications. The objective is to design a layout which minimizes the expected flow cost under the assumption of stochastic demand. Therefore, the resulting layout-flow model is a stochastic mixed integer model which is NP-hard and heuristics are developed to solve it. (Notice that if the requirements were varying, but deterministic, and designs could not easily be changed, we would end up with the same model if we wished to minimize average cost.)

Our paper uses the layout-flow model to *analyze optimal designs* for factories facing random demand levels. We wish to understand what characterizes optimal designs, both since that is useful information in its own right, and because it can be used to develop heuristics. This is a special case of the layout-flow model since one of the uncertain aspects, i.e. the sequence of machine types, is fixed leaving the only uncertainty from the change in demand levels. In real application, this represents cases in which factories can only produce a single type or one fixed mixture of products but the amount of demand varies deterministically or stochastically over time. Hence, the model used in this paper is a special case of the layout-flow model which is a stochastic mixed integer programming model.

Using CPLEX 11.0 and GUROBI 3.0 on nine-machine instances of the model, optimal solutions can be found by exact methods. One of the features found is that in some cases the optimal solution for maximal demand (which equals the maximal production capacity in our setup) is the same as the optimal solution for the stochastic demand case. Such a feature suggests that, instead of solving stochastic FLPs, some stochastic optimal solutions can be obtained by solving deterministic problems for maximal demands. This feature can be applied to develop a heuristic to obtain good solutions of FLPs for factories which face uncertain demand levels.

The general idea of the heuristic is to start by formulating a special network flow model for the maximal demand case. The model is used to find a set of machine-to-machine flows. Using these flows, rather than the original flows which were between pairs of machine *types*, the layout-flow problem can be transformed to the classical QAP model. In total, our heuristic transforms the stochastic layout-flow model into a deterministic QAP model.

The main benefit of this approach is that we can find good solutions to the stochastic facility location model whenever we can solve the classical deterministic QAP. Although that is in its own right a very hard problem, bringing the computing times for the stochastic case at par with the computing times for the corresponding deterministic case is a major step forward. Preliminary computational experience shows that the results, measured in terms of optimality gaps for the stochastic case (when the gap can be calculated), are very strong.

We shall present the characterizations of the optimal designs, the resulting heuristics, as well as detailed computational results.

References

1. A. Drira, H. Pierreval, and S. Hajri-Gabouj, Facility layout problems: A survey. *Annual Reviews in Control*, 31, 255-267 (2007).
2. A. Kusiak, and S.S. Heragu, The facility layout problem. *European Journal of Operational Research*, 29, 229-251 (1987).
3. R.E. Burkard, and K.-H. Stratmann, Numerical investigations on quadratic assignment problems. *Naval research logistics quarterly*, 25, 129-148 (1978).
4. S.P. Singh, and R.R.K. Sharma, A review of different approaches to facility layout problems. *International Journal of Advanced Manufacturing Technology*, 30, 425-433 (2006).
5. S. Benjaafar, and M. Sheikhzadeh, Design of flexible plant layouts. *IIE transactions*, 32, 309-322 (2000).

Deterministic and stochastic models for investment in different technologies for electricity production in the long period

Maria Teresa Vespucci · Marida Bertocchi · Mario Innorta · Stefano Zigrino

Abstract We present stochastic models for determining the optimal mix of different technologies for electricity generation, ranging from coal, nuclear and combined cycle gas turbine to hydroelectric, wind and photovoltaic, taking into account the existing plants, the cost of investment in new plants, the maintenance cost, the purchase and sales of CO_2 emission trading certificates and green certificates to satisfy regulatory requirements. We assume the point of view of price-taker producer. The stochasticity is related to the future fuel prices and to the future price of CO_2 emissions permits: we generate scenarios for the fuel prices and only one scenario for CO_2 emission permits. The stochasticity appears in the expected costs and the probability that the total cost do not overcome a specific threshold is taken into account by considering CVaR risk measure.

1 Introduction

The incremental selection of energy generation capacity over a long term planning horizon is of great importance for energy planners. It can be considered from two different viewpoints:

1. the single power producer's (GenCo's) perspective,
2. the system operator's perspective.

For problem (1) a mathematical model is needed to determine the optimal expansion plan subject to all the relevant factors (regulations and prices) with the aim of finding an

Maria Teresa Vespucci
Department of Information Technology and Mathematical Methods, University of Bergamo, Via Marconi 5a,
24044 Dalmine (BG), Italy. E-mail: maria-teresa.vespucci@unibg.it

Marida Bertocchi
Department of Mathematics, Statistics, Computer Science and Applications, Via dei Caniana 2, 24127 Bergamo,
Italy. E-mail: marida.bertocchi@unibg.it

Mario Innorta
Department of Information Technology and Mathematical Methods, University of Bergamo, Via Marconi 5a,
24044 Dalmine (BG), Italy. E-mail: mario.innorta@fastwebnet.it

Stefano Zigrino
Department of Information Technology and Mathematical Methods, University of Bergamo, Via Marconi 5a,
24044 Dalmine (BG), Italy. E-mail: stefano.zigrino@unibg.it

optimal trade-off between profit and risk. For case (2) a zonal representation of the production and transmission system must be used, in order to determine GenCo's expansion plans (where and when), taking into account the impact on the transmission network, aiming at minimizing the system cost of satisfying load. There are various references in the literature referring to problems (1) and (2). For problem (1) we mention the papers [1] and [4]. Paper [4] is the base of our model. We differ from it both for dealing differently with the fixed and variable costs and for using a stochastic programming based model. More details about our model can be found in report [11]. Reference [3] is particularly useful in evaluating the use of different risk measures. For case (2) we mention the relevant contributions by [5], [7], [10], [2], [6].

In this paper we deal with problem (1), i.e. we want to determine the optimal generation expansion plan of a single generation company (GenCo) over a long term planning horizon consisting in I years (generally, 30 or more years). In section 2 we describe the model, in Section 3 the scenario generation is discussed, Section 4 discusses the numerical results and the conclusions follow.

2 The stochastic model

The main factors affecting power producers decisions can be summarized in the following:

- regulatory constraints
- uncertainty of prices (fuels, CO_2 emission permits and green certificates)
- characteristics of available technologies
- sites where plants can be located
- budget.

In the model we suppose that sites for locating new plants of each technology are already available, thus we want to determine the number of sites for each technology.

Let J^{ET} denote the set of existing thermal plants and J^{ER} denote the set of existing renewable plants, J^{NT} denote the set of candidate thermal plants and J^{NR} denote the set of candidate renewable plants. Let $J = J^{ET} \cup J^{ER} \cup J^{NT} \cup J^{NR}$. The model decision variables are the following:

- Real variables: let $E_{j,i}$ be the electricity produced by a plant of technology j in year i ; the types of available technologies and their characteristics are illustrated in Tables 1 and 2.
- Integer variables: let $w_{j,i}$ be the number of plants of technology j whose construction has to be started in year i .

In our model the capacity expansion is not a continuous variable but takes into account the size of each candidate plant.

The main goal of the model is to determine the evolution of the optimal investment along the planning horizon of 30 years.

The total number of new plants for technology j , that can be built in the planning horizon, is bounded above by the number of available sites $\bar{Z}_j$:

$$\sum_{i=1}^I w_{j,i} \leq \bar{Z}_j. \quad (1)$$

Table 1 Technologies available for new plants and their characteristics

	installed capacity (MW)	construction time (years)	industrial life (years)
Coal	600	4	25
CCGT	800	2	25
Nuclear	1200	7	50
Biomass	20	1	15
Wind (non progr)	100	1	20
Geothermal	40	3	15
Mini hydro (partly progr.)	1	1	40

Table 2 Technologies available for new plants and their characteristics (cont.)

	CO2 emission rate (t/GWh)	operating hours per year (hours)	Current fuel cost (euro/MWh)	Investment cost (Meuro/MW)
Coal	338	7884	23	1
CCGT	200	7884	45	0.47
Nuclear	—	7884	5.5	2.6
Biomass	—	6000	30	2.2
Wind (non progr)	—	2215	—	1.75
Geothermal	—	7500	—	3.5
Mini hydro (partly progr.)	—	3500	—	3

The maximum production of new plants in year i depends on the installed capacity P_j , the maximum number H_j of operating hours of plant j and on the total number $W_{j,i}$ of operative plants j available for production in year i

$$E_{j,i} \leq (1 - \nu_j) \cdot P_j \cdot H_j \cdot W_{j,i} \quad (2)$$

where ν_j represents the losses and

$$W_{j,i} = \sum_{i-(S_j+L_j-1) \leq k \leq i-S_j} w_{j,k} \quad (3)$$

that is the sum of plants j , whose construction started in year k , provided in year i construction (S_j describes the length in years for construction) is completed and industrial life L_j is not ended.

The maximum production of existing plants in year i depends on the installed capacity P_j , the maximum number H_j of operating hours of plant j and on the residual life $\hat{L}_j$.

$$E_{j,i} = \bar{E}_{j,i} = \begin{cases} (1 - \nu_j) \cdot P_j \cdot H_j & \text{if } i \leq \hat{L}_j \\ 0 & \text{if } i > \hat{L}_j. \end{cases} \quad (4)$$

The green certificates (GC) to be bought or sold are defined as

$$G_i = \eta_i \cdot \sum_{j \in J} E_{j,i} - \sum_{j \in J^{ER} \cup J^{NR}} E_{j,i} \quad (5)$$

where the first term is the total production in year i , η_i is the ratio between renewable energy and total energy and the second term is total production using renewable in year i .

The CO_2 emission permits to be bought or sold are given by

$$Q_i = \sum_{j \in J^{ET} \cup J^{NT}} \theta_j \cdot E_{j,i} - A_i^{CO_2} \quad (6)$$

where θ_j is the CO_2 emission rate, the first term represents the total emission in year i (existing and new thermal plants) and the last term represents the CO_2 emission allowance.

The budget constraint is given by

$$\sum_{i=1}^I \frac{1}{(1+r)^{i-1}} \cdot \left(\sum_{j \in J^{NT} \cup J^{NR}} I_j^a \cdot W_{j,i} \right) \leq B \quad (7)$$

where r is the discount rate, the quantity in parenthesis represents the annual investment cost, I_j^a represents the annual investment cost for technology j and B the available budget.

The GENCO's initial market share is given and an hypothesis is made on the evolution of the GENCO's market share along the planning horizon: $\bar{M}_i$ represents the GenCo's market share in year i . In this way we take into account the fact that the producer cannot reduce his own bid prices below a certain level (the marginal production cost) and therefore its production will be bounded above by the market share

$$\sum_{j \in J} E_{j,i} \leq \bar{M}_i. \quad (8)$$

The expected profit term F in the objective function takes the following form:

$$\begin{aligned} F = \sum_{i=1}^I \frac{1}{(1+r)^{i-1}} & \left(\pi_i^E \cdot \sum_{j \in J} E_{j,i} - \pi_i^{GC} \cdot G_i - \pi_i^{CO_2} \cdot Q_i + \right. \\ & - \sum_{j \in J^{ER}} (f_j + v_j \cdot E_{j,i}) - \sum_{j \in J^{NR}} [(I_j^a + f_j) \cdot W_{j,i} + v_j \cdot E_{j,i}] + \\ & \left. - \sum_{s \in S} p_s \cdot \left\{ \sum_{j \in J^{ET}} (f_j + v_{j,s} \cdot E_{j,i}) + \sum_{j \in J^{NT}} [(I_j^a + f_j) \cdot W_{j,i} + v_{j,s} \cdot E_{j,i}] \right\} \right) \end{aligned} \quad (9)$$

where $\pi_i^E, \pi_i^{GC}, \pi_i^{CO_2}$ are respectively the prices for electricity (exogenous to the model), green certificates and CO_2 , f_j are fixed costs, v_j are variable costs for renewable, $v_{j,s}$ are variable costs for thermal plants depending on fuels prices and p_s is the probability of scenario s .

For the risk term we consider the conditional value at risk ($CVaR$) of profits.

The CVaR is given (see [8] and [9]) by

$$CVaR = VaR - \frac{1}{\alpha} \cdot \sum_{s \in S} (p_s \cdot d_s) \quad (10)$$

with d_s subject to the following constraints:

$$d_s \geq VaR - F_s, \quad d_s \geq 0. \quad (11)$$

being F_s defined by

$$F_s = \sum_{i=1}^I \frac{1}{(1+r)^{i-1}} \left(\pi_i^E \cdot \sum_{j \in J} E_{j,i} - \pi_i^{GC} \cdot G_i - \pi_i^{CO_2} \cdot Q_i + \right. \\ \left. - \sum_{j \in J^{ER}} (f_j + v_j \cdot E_{j,i}) - \sum_{j \in J^{NR}} [(I_j^a + f_j) \cdot W_{j,i} + v_j \cdot E_{j,i}] + \right. \\ \left. - \sum_{j \in J^{ET}} (f_j + v_{j,s} \cdot E_{j,i}) + \sum_{j \in J^{NT}} [(I_j^a + f_j) \cdot W_{j,i} + v_{j,s} \cdot E_{j,i}] \right) \quad (12)$$

Our objective function becomes $F + \lambda \cdot CVaR$, to be maximized, and λ represents the risk-aversion parameter. $\lambda = 0$ represents maximization of expected profit, while increasing values of λ gives importance to the fact that we want to limit the probability to get a value for profit less than a certain threshold.

3 Scenarios generation

In this paper we consider scenarios for fuel prices (coal, gas, nuclear fuel). Scenarios are built on the ratios "estimated price over current price" of fuels for thermal plants combining in all possible ways the quantities in Tables 3, 4 and 5, giving a total of 512 scenarios

Table 3 COAL (23 Euro/MWh)

ratio	0.85	0.90	1.00	1.30	1.80	2.30	2.70	3.00
prob	0.010	0.050	0.560	0.220	0.109	0.035	0.014	0.002

Table 4 GAS (45 Euro/MWh)

ratio	0.90	0.98	1.00	1.20	1.60	2.20	2.60	3.00
prob	0.005	0.030	0.500	0.300	0.116	0.030	0.014	0.005

Table 5 NUCLEAR FUEL (5.5 Euro/MWh)

ratio	0.80	0.90	1.00	1.40	1.80	2.00	2.50	3.00
prob	0.020	0.090	0.752	0.120	0.010	0.005	0.002	0.001

The supply of gas may in some period be more critical than coal, while the supply of nuclear fuel is always available. These considerations are reflected in price scenarios, leading to more stable prices for nuclear fuel with respect to coal and more volatile price for gas with respect to coal.

We suppose that the price $\pi_i^{CO_2}$ for CO_2 emissions increases of a certain percentage per year which is related to CO_2 allowances, see Figure 2. For green certificates, π_i^{GC}

currently represents the green certificate price; it is already foreseen that this incentive will be substituted by feed-in tariff (or feed-in premium) for renewable sold energy in the Italian market (Decree 3 March 2011).

As far as annual market share for energy production, $\bar{M}_i$ of the GENCO, see Figure 1.

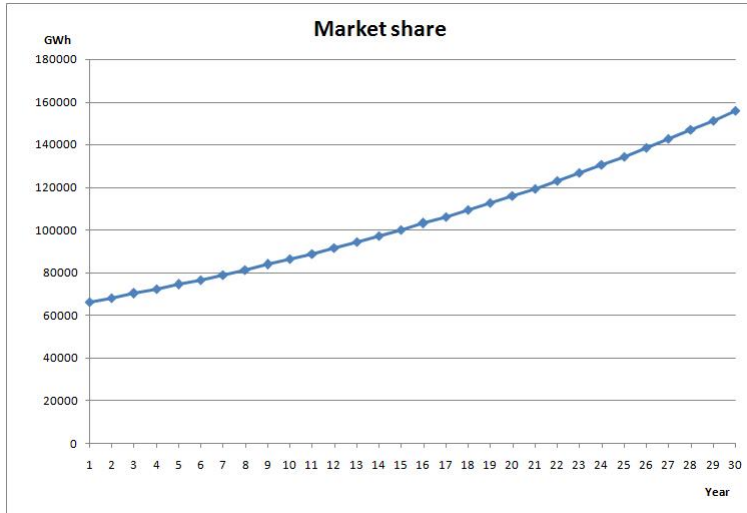


Fig. 1 Scenario for market share.

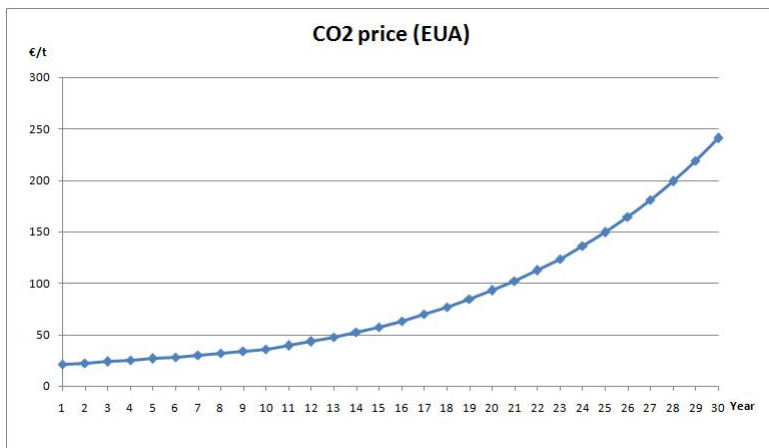


Fig. 2 Scenario for CO_2 price (EUA)

4 Computational results

We consider two stochastic models using 512 different scenarios; both models maximize a linear combination of function given by (9) with the risk measure given by (10), suitably weighted by the risk-aversion parameter λ , subject to

1. constraints (1) to (7) plus constraints (11)
2. constraints (1) to (8) plus constraints (11).

4.1 Case 1. (without market share constraint)

To make a comparison among the two models, we have produced Table 6, where we report the different risk-aversion coefficients, the expected profit and the value of CVaR for medium-large GENCO. The installed electric power for this GENCO is about 9 GW and the current annual energy production is 67,000 GWh. We suppose no constraints on the possibility to increase electric power.

Table 6 Results of stochastic model - Case 1

risk-aversion	Profit	CVaR	VaR	Risk premium
0.01	$1.08 \cdot 10^8$	$-1.62 \cdot 10^8$	$1.53 \cdot 10^7$	6.63
0.13	$8.92 \cdot 10^7$	$4.16 \cdot 10^6$	$2.51 \cdot 10^7$	15.92
1.35	$7.37 \cdot 10^7$	$4.90 \cdot 10^7$	$5.53 \cdot 10^7$	28.72

For different risk-aversion coefficients, it appears the role of CVaR: more risk-aversion corresponds to larger CVaR, larger VaR (it means that our threshold is increasing) and larger risk premium. See Figures 3, 4, 5 of profit distributions for different risk-aversion values.

Figures 6, 7, 8 illustrate the optimal technologies mix for different values of risk-aversion parameter. For risk-aversion equal to 0.01 we get CCGT plants for 98.71%, the rest is in renewable due to their small budget requirement. For risk-aversion equal to 0.13 we have 25% in CCGT plants, 71.66% in coal and the rest in renewable due to their small budget requirement. The choice of coal is motivated by the fact that this technology is less influenced by the price of fuels. For risk-aversion equal to 1.35 we have 50.06% in nuclear plants, 41.93% in coal and the 8.01% rest in renewable.

Total annual energy production given by the optimal decision variables are shown in Figures 9, 10, 11.

4.2 Case 2. (with market share constraint)

To make a comparison among the two models, we have produced Table 7, where we report the different risk-aversion coefficients, the expected profit and the value of CVaR for medium-large GENCO. We suppose that this production may increase by a 3% per year, see Figure 1. We also suppose this evolution is fixed for each of our generated scenarios.

For different risk-aversion coefficients, the behavior is similar to Case 1., i.e more risk-aversion corresponds to larger CVaR, larger VaR (it means that our threshold is increasing) and larger risk premium, see Table 7.

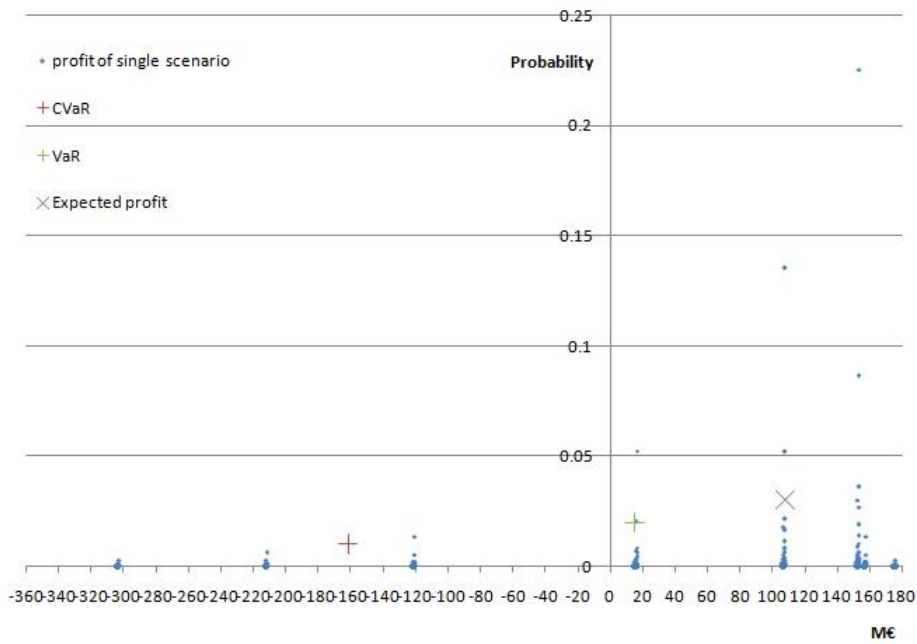


Fig. 3 Case 1, risk-aversion parameter = 0.01: profit distribution.

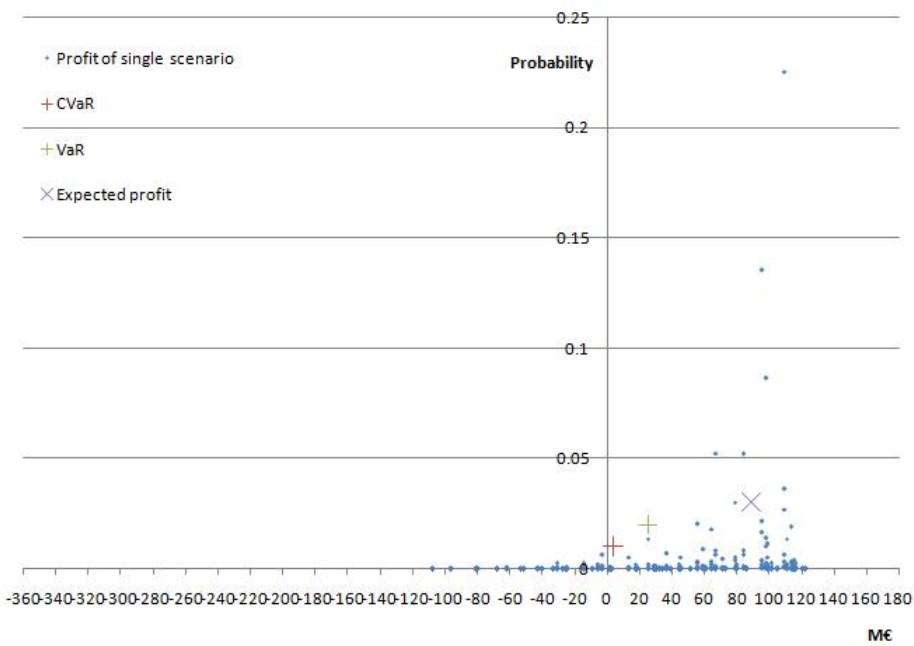


Fig. 4 Case 1, risk-aversion parameter = 0.13: profit distribution.

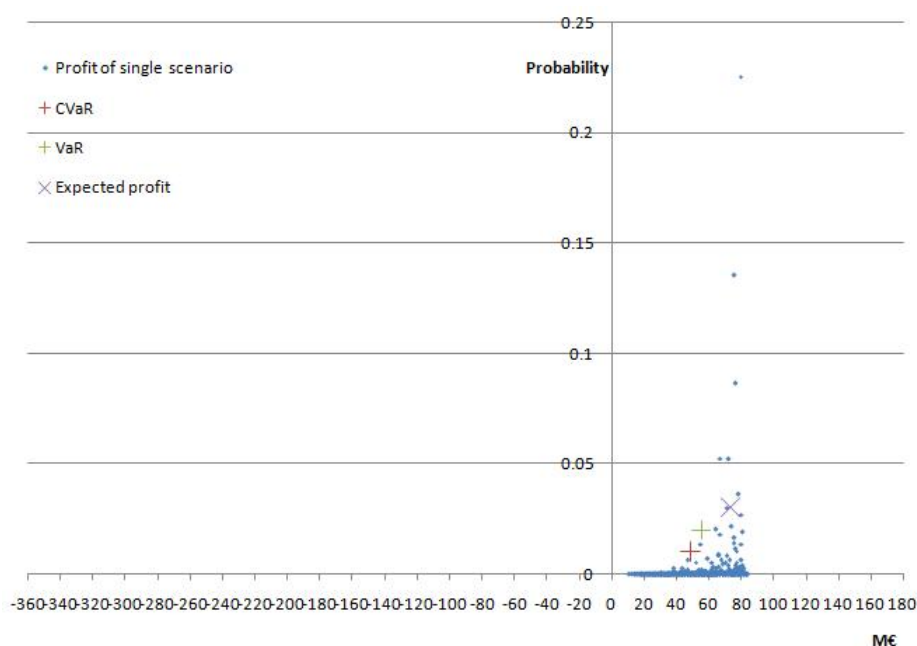


Fig. 5 Case 1, risk-aversion parameter = 1.35: profit distribution.

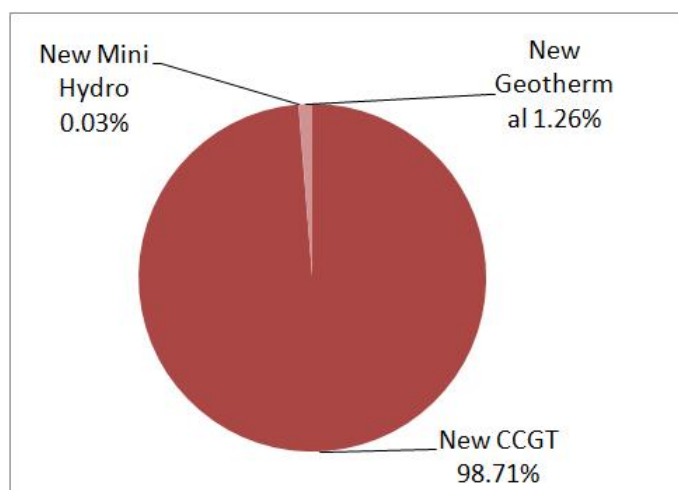


Fig. 6 Case 1, risk-aversion parameter = 0.01: optimal new technologies mix.

Figures 12, 13, 14 show the profit distributions for different risk-aversion values.

Figures 15, 16, 17 illustrate the optimal technology mix for different values of risk-aversion parameter. For risk-aversion equal to 0.01 we have CCGT plants for 31.66%, nuclear plants for 23.13%, wind power plants for 23.78%, biomass plants for 13.78%, geothermal plants for 7.26%. For risk-aversion equal to 0.13 we have 34.81% in coal, 23.08% in nuclear, 20.42% in wind, 14.05% in biomass and the rest 7.24% in geothermal. For risk-

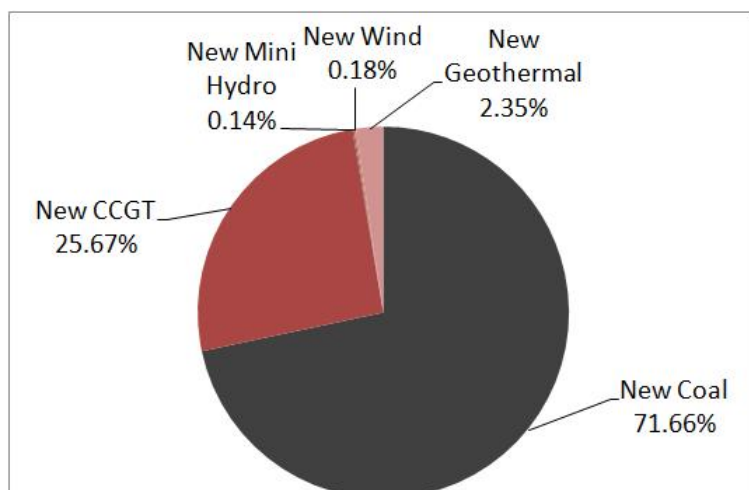


Fig. 7 Case 1, risk-aversion parameter = 0.13: optimal new technologies mix.

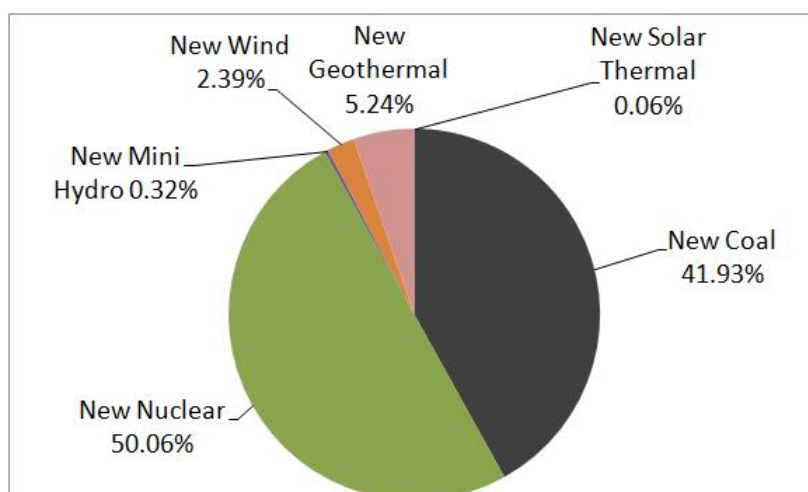


Fig. 8 Case 1, risk-aversion parameter = 1.35: optimal new technologies mix.

Table 7 Results of stochastic model - Case 2

risk-aversion	Profit	$CVaR$	VaR	Risk premium
0.01	$8.63 \cdot 10^7$	$1.71 \cdot 10^7$	$6.13 \cdot 10^7$	20.98
0.13	$8.51 \cdot 10^7$	$3.21 \cdot 10^7$	$5.94 \cdot 10^7$	23.72
1.35	$7.38 \cdot 10^7$	$5.02 \cdot 10^7$	$5.69 \cdot 10^7$	30.47

aversion equal to 1.35 we have 54.68% in nuclear plants, 27.49% in coal and the 11.33% in biomass and 5.72% in geothermal.

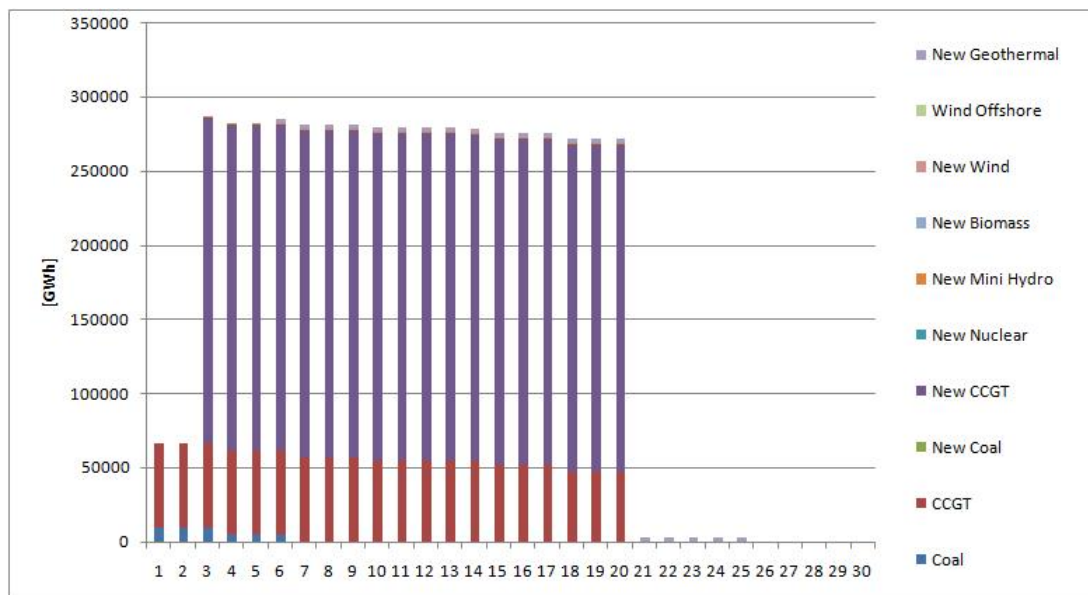


Fig. 9 Case 1, risk-aversion parameter = 0.01: optimal new technologies mix.

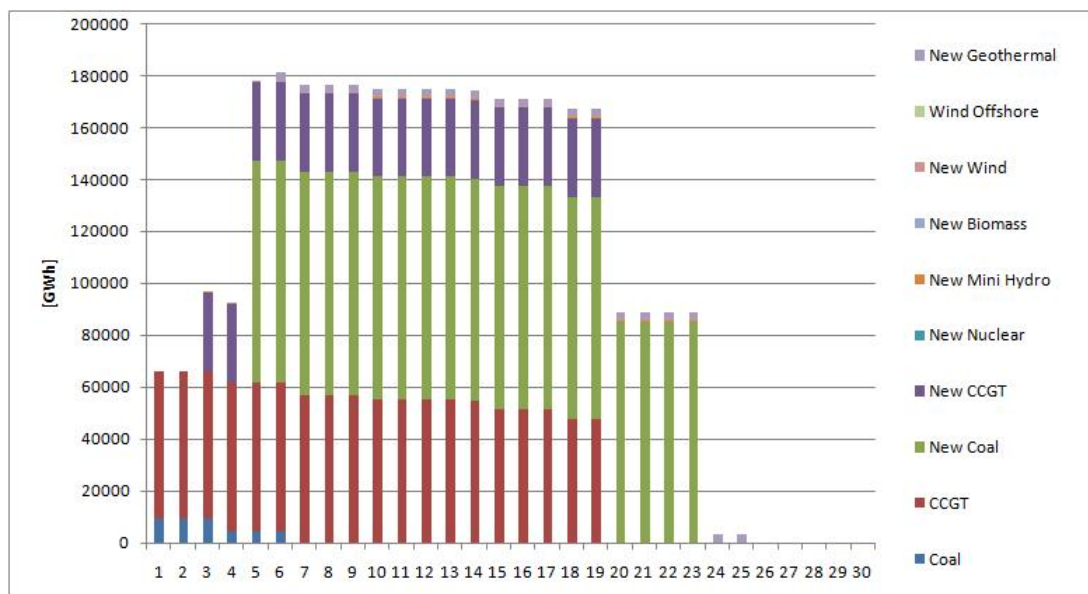


Fig. 10 Case 1, risk-aversion parameter = 0.01: optimal new technologies mix.

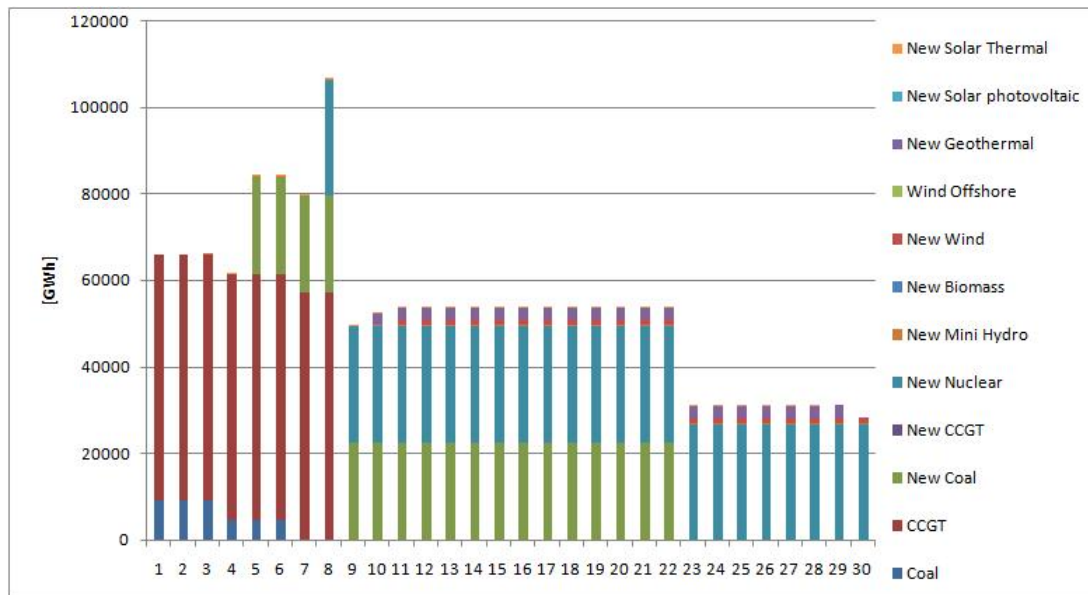


Fig. 11 Case 1, risk-aversion parameter = 0.01: optimal new technologies mix.

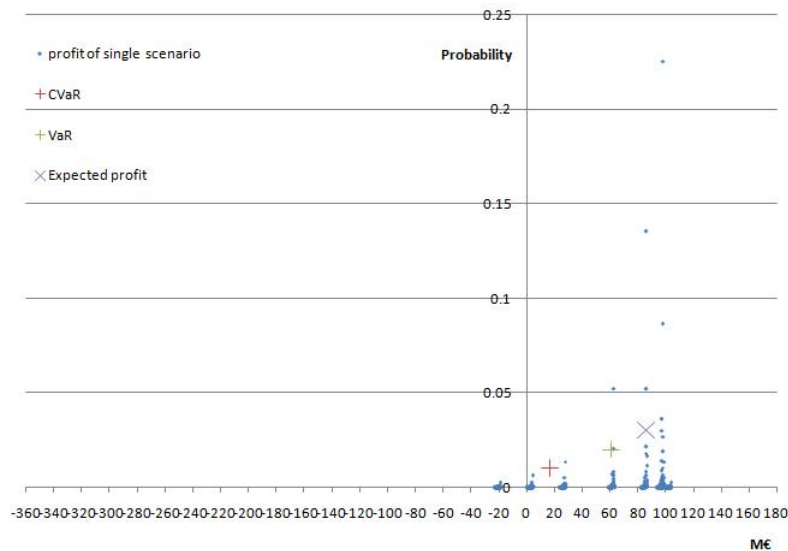


Fig. 12 Case 2, risk-aversion parameter = 0.01: profit distribution.

Total annual energy production given by the optimal decision variables are shown in Figures 18, 19, 20.

Comparing Tables 6 and 7 corresponding respectively to results with and without constraint (8), we observe that, for the same risk-aversion, we get a CVaR larger in Case 2 with respect to Case 1. This means that we are moving towards choices of investment that are more diversified.

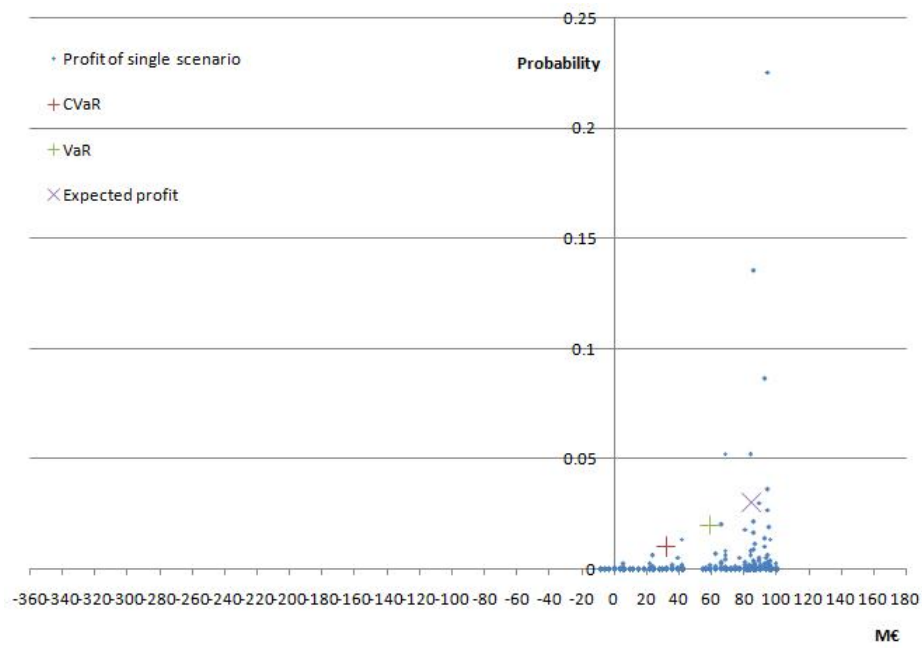


Fig. 13 Case 2, risk-aversion parameter = 0.13: profit distribution.

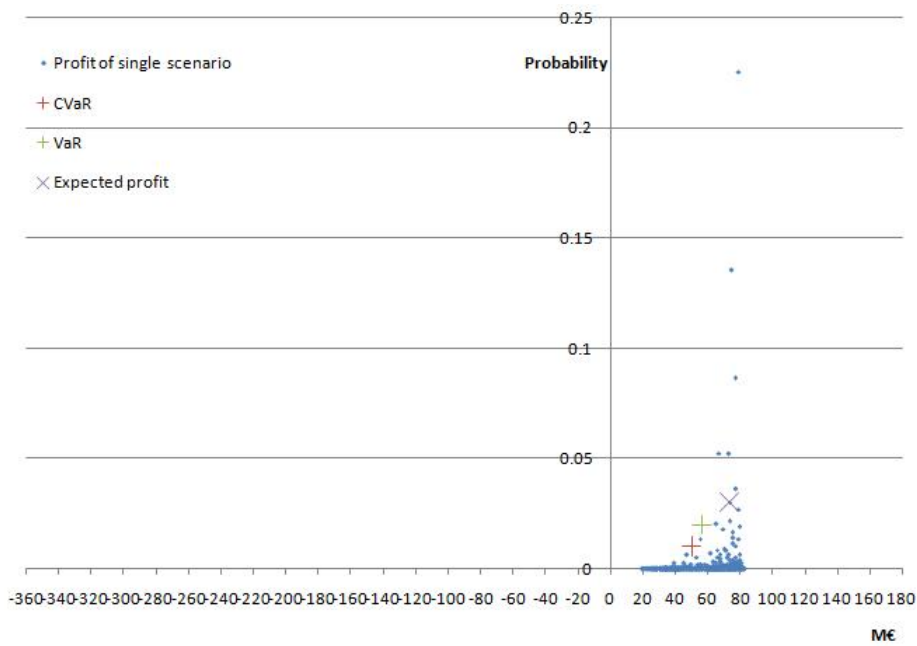


Fig. 14 Case 2, risk-aversion parameter = 1.35: profit distribution.

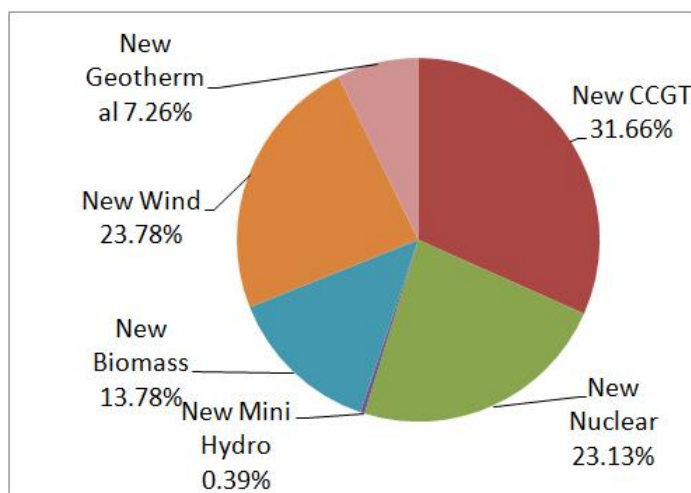


Fig. 15 Case 2, risk-aversion parameter = 0.01: optimal new technologies mix.

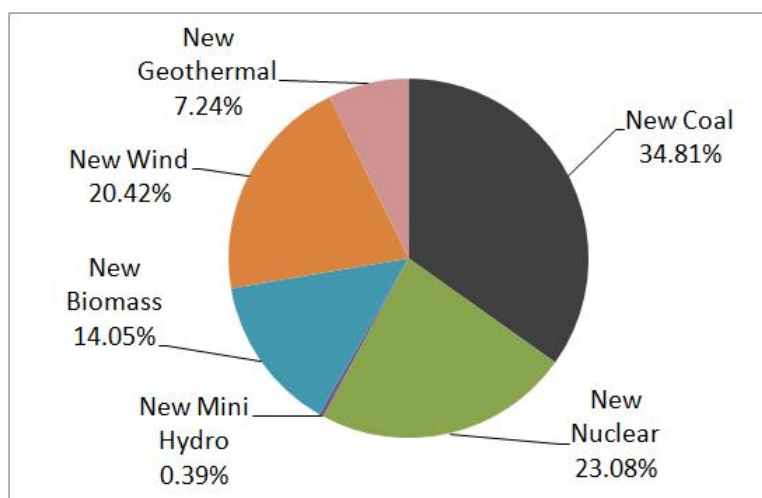


Fig. 16 Case 2, risk-aversion parameter = 0.13: optimal new technologies mix.

5 Conclusion

We adopt a simple scenarios generation combining in all possible ways the 8 scenarios for coal, gas and nuclear fuel for a total of $8^3 = 512$ scenarios. The results from the stochastic multiperiod model show that, taking into account risk, the total expected profit decreases. There is a substantial difference of behavior between the total energy production when we consider or not the limitation on the market quota. The case with market quota is more realistic. Moreover, once the new plants come in operation, there is a period of stable energy production that declines towards the end of the planning period. This may be due to the so-called end of effect, related to the fact that you may have new plants that still have some residual life beyond the end of the planning horizon.

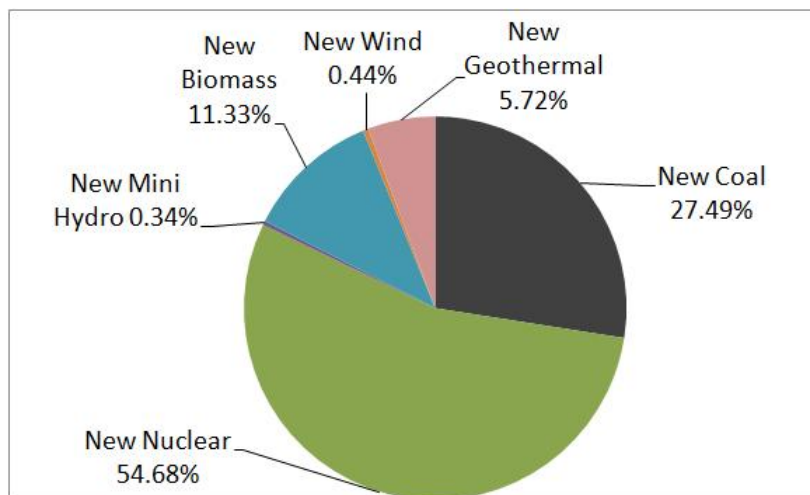


Fig. 17 Case 2, risk-aversion parameter = 1.35: optimal new technologies mix.

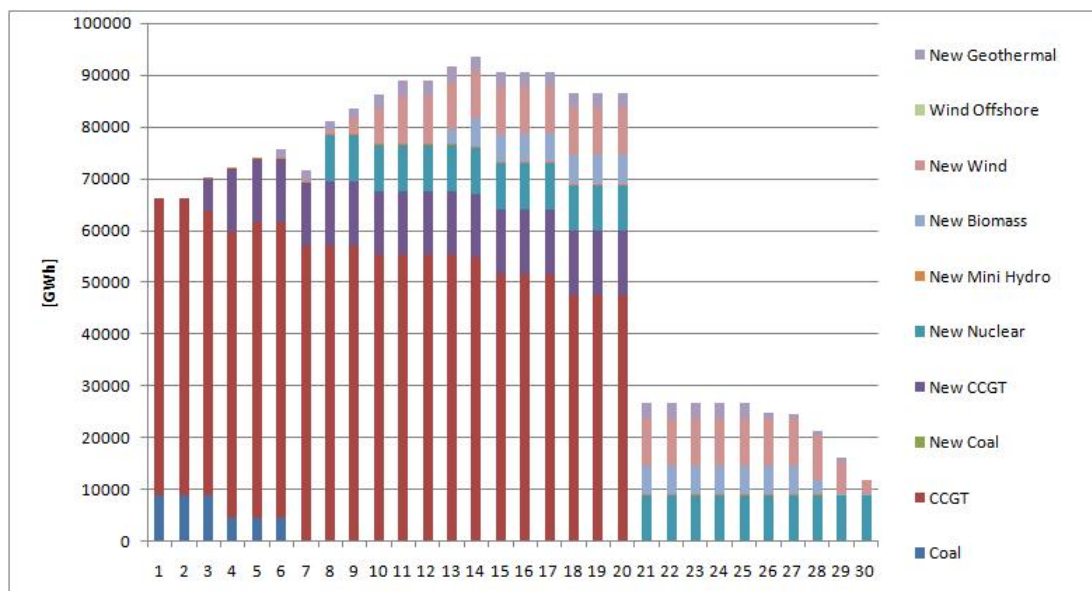


Fig. 18 Case 2, risk-aversion parameter = 0.01: optimal new technologies mix.

Acknowledgements The authors acknowledge the support from the grant by Regione Lombardia "Metodi di integrazione delle fonti energetiche rinnovabili e monitoraggio satellitare dell'impatto ambientale", EN-17, ID 17369.10, and University of Bergamo grants (2010,2011) coordinated by M.Bertocchi and M.T. Vespucci.

References

1. Bjorkvoll T., Fleten S.E., Nowak M.P., Tomasgard A., Wallace S.W., Power generation planning and risk management in a liberalized market, IEEE Porto Power Tech Proceedings, 1, 426-431 (2001)



Fig. 19 Case 2, risk-aversion parameter = 0.13: optimal new technologies mix.

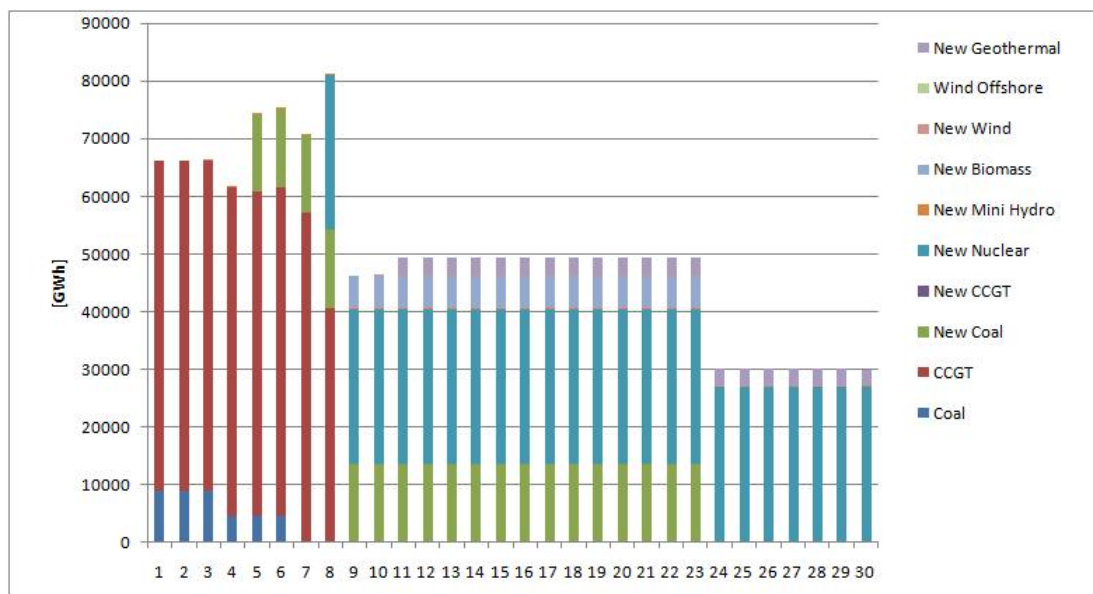


Fig. 20 Case 2, risk-aversion parameter = 1.35: optimal new technologies mix.

2. Botterud A., Mahalik M.R., Veselka T.D., Ryu H.S., Sohn, K.W. Multi-agent simulation of generation expansion in electricity markets, IEEE Power Engineering Society General Meeting Proceedings, Tampa (FL), 1-8 (2007)
3. Conejo A.J., Carrion M., Morales J.M., Decision making under uncertainty in electricity market, International Series in Operations Research and Management Science. Springer Science+Business Media, New York (2010)
4. Genesi C., Marannino P., Montagna M., Rossi S., Siviero I., Desiata, L., Gentile G., Risk management in long term generation planning, 6th Int. Conference on the European Energy Market, 1-6 (2009)
5. Han S., Lee J., Kim T., Park, S., Kim B.H, The development of the generation expansion planning system using multi-criteria decision making rule, The International Conference on Electrical Engineering (2009)
6. Hariyanto, N., Nurdin, M., Haroen, Y., Machbub, C., Decentralized and simultaneous generation and transmission expansion planning through cooperative game theory, International Journal of Electrical Engineering and Informatics, 1(2), 149-164 (2009)
7. Moghdass-Tafreshi S.M., Shayanfar H.A., Saliminia Lahiji A., Rabiee A., Aghaei J., Generation expansion planning in Pool market: a hybrid modified game theory and particle swarm optimization, Energy Conversion and Management, 52, 1512-1519 (2011)
8. Rockafellar R.T., Uryasev S., Optimization of conditional value-at-risk, Journal of Risk, 2, 21-41 (2000)
9. Rockafellar R.T., Uryasev S., Conditional value-at-risk for general loss distributions, Journal of Banking and finance, 26, 1443-1471 (2002)
10. Roh J.H., Shahidehpour M., Fu Y., Market-based coordination of transmission and generation capacity planning, IEEE Transaction on Power Systems, 22(4), 1406-1419 (2007)
11. Vespucci M.T, Bertocchi M., Innorta I., Zigrino S., Models for the generation expansion problem in the Italian electricity market, Technical Report, University of Bergamo (2011)

Ring Out the Bells

J. Donaldson, B. McCollum, S. Scott

1 Introduction

Campanology is the art or study of bell casting and ringing. For many people this is an interesting and fulfilling hobby. Every year in southern England a campanology event is held which aims to introduce beginners to the hobby as well as allowing more experienced campanologists the opportunity to learn new techniques and visit different church towers.

The event takes place over a four day period (Thursday – Sunday) in which participants are organized into teams for the duration of the event. Each team participates in 6 sessions of bell ringing during which participants are tutored by experienced campanologists with the aim that by the end of the weekend the team will be able to ring the bells independent of the tutors' instruction. Each of the six sessions are held in a different church towers. This gives each team exposure to aspects such as differing construction of towers, different bell weights, rope draught and tension etc.. allowing them to deepen their knowledge of the nuances involved in ringing bells in different settings.

Within bell ringing methods are rung on different numbers of bells e.g. double methods require 5 method bells and a cover bell; minor methods require 6 bells; triple methods require 7 method bells and a cover bell. Due to this not all church towers are suitable for certain styles of ringing.

This makes the scheduling of this event difficult as not only do considerations need to be made for the suitability of each church tower, but also, due to the dispersion of church towers among the region, minimal movement of the teams between each of their sessions would be preferred. Additionally on top of this, considerations could be made for issues such as local amenities, nearby accommodation, annoyance of local residents and access to the towers themselves.

J R Donaldson
School of Electronics, Electrical Engineering and Computer Science, Queen's University,
Belfast
E-mail: jdonaldson10@qub.ac.uk

B G C McCollum
School of Electronics, Electrical Engineering and Computer Science, Queen's University,
Belfast
E-mail: b.mccollum@qub.ac.uk

S. Scott
School of Electronics, Electrical Engineering and Computer Science, Queen's University,
Belfast
E-mail: ns.scott@qub.ac.uk

2 Background Research

The complex nature of bell ringing is highlighted in several research papers which explore the mathematics of the hobby [1, 2]. Reading these reinforces the need for accurate measurements by which the suitability of each bell tower may be judged. In regards to the timetabling aspect of the problem, generally this type of problem is regarded as a routing problem. However, beyond route finding there are many examples from literature and the real world that fit into the same group of problems [9]. Many approaches to the topic of timetabling have been presented ranging from simply guided heuristic searches [3] to complex meta heuristic methods such as tabu search [4, 5] and simulated annealing [6].

3 Methodology

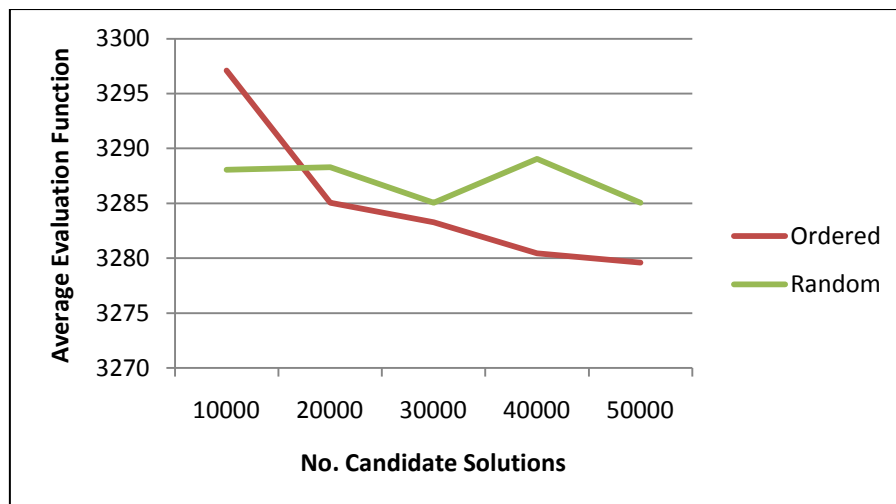
The initial approach to formulating a solution was to model this as a graph coloring problem. The generation of proposed solutions is guided through the use of an ordering heuristic that orders the teams based on the difficulty required in scheduling, such as a large number of team members or teams consisting entirely of beginners. Following the generation of a pool of feasible candidate solutions, an evaluation function is applied to each timetable. This provides an accurate representation to allow comparison between candidate solutions.

To enable quick and accurate calculation of the distances between bell towers, a distance matrix containing the road distance between each tower is pre-computed using Google Maps. The suitability of towers is calculated by comparing the rated skill level of the members of a team the weight of the bells in that tower.

The heuristic used in the evaluation function is comprised of a summation of statistical measures derived from the candidate solution. The aim of this evaluation function is to combine measures for both the suitability of the towers assigned to teams based on the ringing style that team specializes in and also to minimize the distance moved between sessions. Additionally the evaluation function aims to favor solutions that have an even spread of values across the entire candidate solution. The main downfall of a summation of values such as this is that if one value in the summation is abnormally high or low then it could disproportionately affect the entire value of the summation. To alleviate this issue each individual summation value is weighted in an attempt to ensure that each part of the summation carries equal value. These weights were calculated through analysis of approx. 1,000,000 feasible candidate solutions. Subsequently the evaluation function is as follows:

4 Results

For comparison the developed ordering heuristic was compared to a heuristic that generates random, feasible solutions. What was found was that on average, the greater the number of test solutions generated the better performing the ordered heuristic was as shown in the figure below.



It was also discovered that across all testing the ordered heuristic was approx 15% faster in generating the same amount of candidate solutions as the random heuristic.

This shows that it is possible to create an ordering heuristic that will effectively produce timetables that meet the needs of multiobjective problems. However, for this particular problem and implementation, the heuristic requires the production of many candidate solutions. In real world implementations this could prove problematic if the generation of the time table/schedule is time sensitive.

In regards to the best candidate solutions generated, the ordered heuristic 80% of cases and it produced the 5 of the top 10 candidate solutions 95% of the time. This shows the best solutions generated by the ordering heuristic are very much better than the average performance of the random heuristic.

5 Future Research Directions

This paper describes a real world model of a scheduling problem. The solution implemented currently generates good quality initial solutions and, even given the issues with weighted summations as a means to evaluate multiple objectives, still provides an effective way to select the best candidate solution.

Naturally the next obvious avenue for exploration would be that of meta-heuristic search methods in way of improving upon these initial solutions. Introduction of an improvement phase to the construction of candidate solutions may also allow us to reduce the time required in generation good quality workable solutions. Due to its success in other related areas, we intend to investigate the Great Deluge [7, 8] in this respect. A full implementation and results of both the construction and improvement phase will be presented at the conference.

Future directions of the research include a full multi-objective representation and comparison of the construction and improvement approaches with other related scheduling problems e.g. examination and course timetabling.

References

1. Arthur T. White, Treble dodging minor methods: ringing the cosets, on six bells, Discrete Mathematics Volume 122, Issues 1-3, 15 November 1993, Pages 307-323
2. Chapter 19 Change ringing, North-Holland Mathematics Studies Volume 188, 2001, Pages 295-321

3. E.K. Burke, J.P. Newall, R.F. Weare, A Simple Heuristically Guided Search for the Timetable Problem, International ICSC Symposium on Engineering of Intelligent Systems 1998.
4. Daniel Costa, A tabu search algorithm for computing an operational timetable, European Journal of Operational Research Volume 76, Issue 1, 6 July 1994, Pages 98-110
5. L. Di Gaspero, A. Schaerf, Tabu Search Techniques for Examination Timetabling, Practice and Theory of Automated Timetabling III, Lecture Notes in Computer Science, 2001.
6. J. M. Thompson, K. A. Dowsland, Variants of simulated annealing for the examination timetabling problem, Annals of Operations Research, Volume 63, Number 1
7. B. McCollum, P. J. McMullan, A. J. Parkes, E.K. Burke, An Extended Great Deluge Approach to the Examination Timetabling Problem, submitted to MISTA, Dublin 09
8. McMullan, P, An Extended Implementation of the Great Deluge Algorithm for Course Timetabling, Lecture Notes in Computer Science, Springer, Vol 4487, pages 538-545, 2007
9. A General Vehicle Routing Problem, European Journal of Operational Research, Volume 191, Issue 3, 16 December 2008, Pages 650-660 Asvin Goel, Volker Gruhn

A Hyper-Heuristic Approach to Solving School Timetabling Problems

Nelishia Pillay

Abstract Hyper-heuristics have proven to be an effective means of obtaining generalized solutions to optimization problems. One such domain in which they have been a success is educational timetabling. However, all the research in this area has focused on university course and examination timetabling. This paper is amongst the first to examine the use of hyper-heuristics to solve the school timetabling problem. The study presented extends previous work investigating the use of an evolutionary algorithm hyper-heuristic (EA-HH) to solve an artificially created problem from the Abramson problem set [1]. This study takes this research further by using the EA-HH to solve the school timetabling problem for a South African primary school. A comparison of the findings of both studies is presented.

1 Introduction

The school timetabling problem involves allocating class, teacher and venue tuples to timetable periods so as to meet the hard constraints of the problem and minimize the soft constraints. A timetable meeting the hard constraints is said to be feasible. The hard and soft constraints vary drastically from school to school [4]. Examples of hard constraints include no class, teacher or venue being scheduled more than once in a particular period. Soft constraints include teacher preferences, period preference for teaching certain subjects. e.g. Mathematics, and consecutive periods for some subjects, just to name a few. Various methods have been used to solve the school timetabling problem, namely, simulated annealing, tabu search, constraint programming, integer programming, genetic algorithms, tiling algorithms and graph theory methods, amongst others [4]. However, it appears that hyper-heuristic approaches have not been applied to this domain.

Hyper-heuristics are methods that generate or choose heuristics or combinations of heuristics to solve an optimization problem and aim at providing a generalized solution to the problem ([2], [3], [6]). A hyper-heuristic can be selection or generation based and the low-level heuristics can be perturbative or constructive [3]. The study presented in this paper implements an evolutionary algorithm hyper-heuristic (EA-HH) to search a space of heuristic combinations comprised of low-level construction heuristics for the school timetabling problem. Based on the proposed hyper-heuristic classification in [3], the hyper-heuristic implemented in this study is a selection, offline learning hyper-heuristic. In an initial study to investigate the use of an evolutionary algorithm hyper-heuristic for solving the school

Nelishia Pillay
School of Computer Science, University of KwaZulu-Natal,
KwaZulu-Natal, South Africa
E-mail: pillayn32@ukzn.ac.za

timetabling problem, the EA-HH was applied to an artificial problem [1] from the Abramson problem set [5]. The study presented in this paper extends this work by using the EA-HH to solve a real-world school timetabling problem. The paper compares the performance of the EA-HH in solving both these problems.

The following section describes both the school timetabling problems. Section 3 presents the EA-HH. The performance of the EA-HH on both the problems is discussed in section 4. The findings of the study and future extensions of this work are summarized in section 5.

2 School Timetabling Problems

The EA-HH hyper-heuristic has been applied to two school timetabling problems. The first is one of the “hard” artificial school timetabling problems made available by Smith and Abramson [7]. This problem requires the allocation of class, teacher and venue tuples to timetable slots. The problem involves 4 classes, 4 teachers, 4 venues, 30 periods and 120 requirements. The problem has two hard constraints. Firstly, all requirements must be met, i.e. all class-teacher meetings must be scheduled. Secondly, there must be no clashes, i.e. each class, teacher and venue must not be scheduled more than once in a timetable period.

The second problem is a real-world school timetabling problem requiring a timetable to be constructed for a South African primary school. The South African schooling system is divided into primary and secondary school education. Primary schools usually cater for grades 1 to 7 and high schools for grades 8 to 12. There are 16 classes, 19 teachers and 16 subjects. The problem generally does not include venue allocation as the grades are allocated to fixed classrooms. Some classes have to move for lessons requiring specialized venues, namely, physical education and technology. The school week is comprised of 5 days of 11 periods per week. Each grade has a different number of periods. Grades 1 and 2 have 9 periods on all days of the week. Grade 3 classes have 11 periods on a Monday, 9 periods on a Friday and 10 periods the rest of the week. Grades 4 to 7 have 11 periods Monday through to Thursday and 9 periods on a Friday. The hard constraints for the problem are: all problem requirements must be met, i.e. each class-teacher meeting must be scheduled the required number of times and there must be no timetable clashes, i.e. each teacher and class must only be scheduled once in a timetable period. The soft constraints of the problem are: period preferences - certain subjects must be scheduled during a particular range of periods, e.g. Mathematics must be scheduled during the first six periods of the day; double periods - some subjects must be scheduled in a number of double periods, i.e. two consecutive periods. There are a total of 802 requirements, 90 period preferences and 175 double periods. The following section describes the EA-HH used to solve the school timetabling problems.

3 The Evolutionary Algorithm Hyper-Heuristic

This section describes the evolutionary algorithm hyper-heuristic. The evolutionary algorithm employs the generational control model. An initial population is firstly created. The initial population is then iteratively refined on successive generations via the processes of evaluation, selection, and recombination. Each of these phases is described below.

3.1 Initial population generation

Each element of the population is a string of characters representing low-level construction heuristics for the school timetabling problem. Each heuristic combination is composed of two or more of the following low-level construction heuristics: largest degree(l), i.e. the class-teacher tuple with the largest number of requirements, i.e. class-teacher meetings is given preference; saturation degree(s) which schedules the class-teacher tuple with the least number of feasible timetable periods, i.e. slots that will not result in a class or teacher clash if the tuple

is allocated to it, at the current point of construction first; double degree(d) gives preference to class-teacher tuples requiring double periods or more than one teacher with tuples requiring both a double period and two teachers given preference over tuples requiring only a double period or two teachers; period preference degree(p) which schedules class-teacher tuples involving subjects that have specific period preferences first.

A class-teacher tuple represents a meeting between a teacher and class to be taught a specific subject, i.e. a lesson. An example of an individual is *lssdp*. In this case the first class-teacher tuple will be scheduled using largest degree, the next two with saturation degree, the fourth with double degree and the fifth allocation with period preference degree. The length of each heuristic combination is randomly chosen to be in the range of two and the number of class-teacher tuples to be allocated. If the length of the heuristic combination is less than the number of tuples to be allocated, the heuristics are applied again in order from the first heuristic in the string. If the heuristic combination is longer than the number of tuples to be scheduled, only the first n heuristics are applied, where n is equal to the number of tuples to be created. Each heuristic combination is created by randomly choosing the character, corresponding to a particular low-level heuristic, to occur at each position of the string. The number of elements of the population remains the same from one generation to the next and is set at the beginning of a run. This value is a genetic parameter and the most appropriate value is problem dependant.

3.2 Evaluation and selection

Each element of the population is assigned a fitness measure. This fitness is calculated by using the heuristic combination to construct a timetable for the respective school timetabling problem. The fitness is a function of the hard and soft constraint cost of the constructed timetable. The fitness is used by the selection method to select parents for the next generation.

Tournament selection is used to choose parents for each generation. A number of individuals, referred to as the tournament, are randomly chosen from the population. The fittest individual in the tournament is returned as a parent. The size of the tournament is problem dependant. The tournament selection is called up each time a parent is needed to create an offspring. Selection is with replacement, thus an individual may be chosen to be a parent more than once. Genetic operators are applied to the chosen parent/s to create offspring. The following section describes the mutation and crossover operators.

3.3 Genetic operators

The mutation and crossover genetic operators are applied to copies of the parents to create the offspring of the next generation. The application rate of each genetic operator is problem dependant.

The mutation operator replaces the low-level heuristic at the randomly selected mutation point in the heuristic combination with a randomly chosen low-level heuristic. For example, suppose that 3 is chosen as the mutation point in *lssdp* and p is the randomly selected low-level heuristic. The corresponding offspring is *lspdp*. The mutation point is chosen to be in the range of the number of tuples to be scheduled.

The crossover operator selects crossover points in copies of two parents. The parents are "crossed over" at the crossover points to produce two offspring. Crossover points are chosen to be in the range of the number of tuples to be scheduled. Trial runs have shown that returning the fitter of both offspring is more effective than returning both offspring. Suppose that *pddsl* and *llsssl* are selected parents and 2 and 3 are the crossover points respectively. The corresponding offspring are *psssl* and *liddsl*. The fitter of the two offspring is returned as the result of the operation.

Both the mutation and crossover operators have also been implemented with hill-climbing. In the case of the mutation operator the operation is performed until an offspring that is fitter than the parent is produced. In order to reduce runtimes and prevent premature convergence a limit is set on the number of attempts at producing a fitter offspring. In the case of the crossover operator the operation is repeated until a single offspring that is fitter than at least one of the parents is produced. Again a limit is set on the number of attempts to prevent premature convergence and long runtimes.

4 Results and Discussion

The EA-HH has been applied to the school timetabling problems presented in section 2. The EA-HH was implemented in Java 1.6.2 and all simulations were run on Intel Core 2 Duo processor with Windows XP. All simulations were given a maximum runtime of 2 hours. Ten runs, each using a different seed for the random number generator, were performed for each problem. The genetic parameter values and heuristics used for each of the problems are listed in Table 1. These values were obtained empirically by performing trial runs.

Table 1: Genetic Parameter Values and Heuristics for Problem 1 and Problem 2

	Problem 1	Problem 2
Number of generations	100	50
Population size	500	500
Mutation (with hill-climbing) rate	0.1	0.5
Crossover (with hill-climbing) rate	0.9	0.5
Tournament size	10	10
Number of mutation attempts	30	-
Number of crossover attempts	50	-
Low-level heuristics	Largest degree Saturation degree	Largest degree Saturation degree Double degree Period preference degree

As can be seen from Table 4 different parameter values and heuristics were needed for each of the problems. For the first problem the use of genetic operators with hill-climbing produced better results. However, the use of hill-climbing in solving Problem 2 did not improve the performance of the EA-HH. In previous work [11] different low-level heuristics were evaluated for solving Problem 1, namely, largest degree, saturation degree, teacher degree (gives the tuple with the teacher with the largest number of requirements priority) and class degree (gives the tuple with the class with the largest number of requirements priority). The EA-HH was found to perform the best with combinations of the largest degree and saturation degree heuristics. For Problem 2 the use of just these two heuristics was not sufficient and additional heuristics that cater for the soft constraints were also needed. The fitness function for Problem 1 was the number of hard constraint violations. For Problem 2 both hard constraints and soft constraints had to be taken into consideration. Three fitness functions were tested. The first function was the product of the hard constraint violations plus one and the soft constraint violations. The second fitness function was a weighted sum of the hard and soft constraint violations, with the hard constraints violations being weighted more. The third fitness function performed a Pareto comparison of the hard and soft constraint violations, again with the hard constraints being given priority. The third fitness function was found to be the most effective.

The EA-HH produced feasible solutions for both Problem 1 and Problem 2. The heuristic combination with the best fitness for each run, for both of the problems, contained all the low-level heuristics.

The performance of the EA-HH was comparable to that of the Hopfield neural network and simulated annealing and better than the greedy search and Tabu search applied to the same problem in Smith et al. [7]. The best solution found for Problem 2 had a soft constraint cost of 21 with 5 period preference violations and 16 double period violations. The timetable currently being used by the school is firstly generated by an online system. The timetable produced usually does not meet all the constraints and is manually tweaked to meet as many constraints as possible. The current timetable has 4 double period violations and 22 period preference violations with an overall soft constraint cost of 26.

5 Conclusion and Future Work

The study presented in this paper investigated the use of an EA-HH to solve the school timetabling problem. The EA-HH was applied to a “hard” artificial problem and a real-world problem. The EA-HH was able to produce feasible timetables for both problems. The quality of the timetables produced is comparative to other methods applied to the same problems. The study has revealed that the low-level heuristics combined does have an affect on the overall performance of the hyper-heuristic and different low-level heuristics are needed for the different school timetabling problems as the constraints for these problems differ drastically from one school to the next. Furthermore, for some problems incorporating the use of hill-climbing in the genetic operators enhances the performance of the evolutionary algorithm hyper-heuristic. In addition to this the fitness function used also has an affect on the performance of the EA-HH and for real-world problems with various constraints a Pareto comparison is more appropriate. The research presented in this paper has highlighted the potential of hyper-heuristics in solving the school timetabling problem. Future extensions of this work will investigate applying the EA-HH to additional school timetabling problems in an attempt to derive a general model for an evolutionary algorithm hyper-heuristic for school timetabling.

References

1. Beasley J.F., OR Library.<http://people.brunel.ac.uk/mastjjb/jeb/orlib/tableinfo.html>, last accessed 1 February 2010 (2010).
2. Burke E., Hart E., Kendall G., Newall J., Ross P., Schulenburg S., Hyper-Heuristics: An Emerging Direction in Modern Research. Handbook of Metaheuristics, Chapter 16. Kluwer Academic Publishers, 457– 474 (2003).
3. Burke E.K., Hyde M., Kendall G., Ochoa G., Ozcan E., Woodard J., A Classification of Hyper-Heuristic Approaches, in Handbook of Metaheuristics, International Series in Operations Research and Management Science, Volume 146, 449-468 (2010).
4. Pillay N., An Overview of School Timetabling, in proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010), Queens University Belfast, August 2010, 321-335 (2010).
5. Pillay N., A Study into Hyper-Heuristics to Solve the School Timetabling Problem, in proceedings of SAICSIT 2010, 259-264, ACM Press (2010).
6. Ross P., Hyper-heuristics, in Burke E.K., Kendall G. (eds): Search Methodologies: Introductory Tutorials in Optimization and Decision Support Methodologies, Chapter 17. Kluwer, 529 -556(2005) .
7. Smith K.A., Abramson D., Duke D., Hopfield Neural Networks for Timetabling: Formulations, Methods, and Comparative Results, in Computers and Industrial Engineering 44, Pergamon, 283-305 (2003).

Two Phases Parallel Hyperheuristic Approach to the Bus Driver Scheduling Problem

Shi Li • A. Hajjam • J. C. Créput • A. Koukam

1 Introduction

The bus driver scheduling problem involves partitioning the vehicle work into a set of legal driver shifts that should reflect the operator's definition of efficiency [1]. It is an important and complex part of the operational planning in public transport. Since the constraints according to labour rules and requirement differ from country to country, even company to company, the evaluation criteria and objectives may differ as well.

A great deal of research on the solution of this problem has been done since the 1960's. In [2] [3] [4], the authors introduce a variety of approaches to solve this problem. Methods are more and more required to obtain several good solutions in various scenarios with reasonable time. However, the choices of methods for different situations and environments have proven to be fairly unclear and also non-uniform. Therefore, we propose a flexible method based on the hyperheuristic paradigm to solve this problem.

Hyperheuristics are heuristic management methods aiming at a high level of generality in problem solving [6]. The main property of hyperheuristics is that they operate on a set of heuristics instead of operating directly on the search space of the problem [5]. Hyperheuristics have received much attention recently and will likely remain a hot research topic for the near future.

In our work, we are motivated by the desire to provide a more efficient algorithm and good enough quality solutions by using a two phase parallel hyperheuristic called TPPH. One phase aims to choose the low level operators in order to change the current solution. We constantly attempt to search procedures that aspire to find global optimal solutions, and try to

Shi Li

Université de Technologie de Belfort- Montbéliard - Belfort - France

E-mail: Shi.Li@utbm.fr

Amir HAJJAM EL HASSANI

Université de Technologie de Belfort- Montbéliard - Belfort - France

E-mail: Amir.Hajjam@utbm.fr

Jean-Charles Créput

Université de Technologie de Belfort- Montbéliard - Belfort - France

E-mail: Jean-charles.Creput@utbm.fr

Abder Koukam

Université de Technologie de Belfort- Montbéliard - Belfort - France

E-mail: abderrafiaa.koukam@utbm.fr

avoid reaching a local optimum by using some strategies at the same time. However, when these intensification moves prove ineffective, the higher level neighborhoods are applied in the next phase. The second phase is inspired by the large neighborhood search (LNS [7]) to have a set of rules as a type of diversification for overcoming local optimality, and then return the solution to the first phase after termination.

2 The bus driver scheduling problem

Bus driver scheduling problems are a well-known classes of NP-hard problems [8]. To specify the problem, some terms need to be introduced. A *Relief Opportunities* is a time/location pair at which drivers can leave the current vehicle. The work between two consecutive relief opportunities on the same vehicle is called a *piece of work (task)*. The work of a driver in a day is known as a *shift (duty)*. The set covering and set partitioning formulation are the fundamental models in the bus driver scheduling. However, using a local search technique such as Tabu search, the set partitioning model looks better when compared with the set covering model to find admissible solutions quickly [10]. In the formulation adopted here, we will consider our approach based on the set partitioning formulation of the problem. Let $M = \{1, 2, \dots, m\}$ be the set of pieces of work to be covered and $N = \{1, 2, 3, \dots, n\}$ be the set of legal shifts. We can define the problem as the construction of a matrix, where pieces of works appear in lines and shift in columns. For each element $a_{ij} \in \{0, 1\}$, $i \in M$, $j \in N$, $a_{ij} = 1$ indicates that the shift j covers the piece of work i , it is 0 otherwise. The problem is formulated as a set-partitioning problem (SPP) as follows:

$$\text{Minimize } W1 \sum_{j=1}^n c_j x_j + W2 \sum_{j=1}^n x_j \quad (1)$$

$$\text{Subject to } \sum_{j=1}^n a_{ij} x_j = 1, \quad i \in M \quad (2)$$

$$x_j \in \{0, 1\}, \quad j \in N \quad (3)$$

where n is the number of potential shifts; m is the number of pieces of works; c_j is the cost of shift j ; x_j is equal to 1 if shift j is used in the solution and 0 otherwise. $W1$ and $W2$ are weight constants. In order to compare the results, the costs c_j are the same as in [3]. Note that

the term $\sum_{j=1}^n c_j x_j$ is the total cost, whereas $\sum_{j=1}^n x_j$ is the number of shifts and hence of drivers.

3 Two Phases Parallel Hyperheuristic

In this study, a new parallel hyperheuristic approach based on reinforcement learning is proposed. It consists of an iterative framework with two sub-mechanisms: a heuristic selection mechanism which uses a choice function to select a low level heuristic for generating a new (partial or complete) solution in the current iteration, and an acceptance mechanism to decide on the acceptability of the new solution. The parallel principle is the same as in [11]. It consists

of executing many hyperheuristics simultaneously which cooperate during the search by the way of exchanging their best solutions found.

A choice function which can accumulate historical information about the recent performance of each low level heuristic is applied in order to adapt the probability to choose the next low level heuristic at each decision point. In our study, the selection principle is inspired by the Softmax Action Selection in [9]. It is based on a set of rules of the form $\langle \text{state}, \text{action} \rangle$, where the action corresponds to the next low level operator to apply at the decision point search, and where the state is the previous low level operator applied. A weight P_{ij} is associated to each pair $\langle \text{state}, \text{action} \rangle$ to denote the probability value to select the operator j after applying the operator i . In each the parallel procedure, the effective choice of an operator is performed by this selection principle for each hyperheuristic independently.

There are two phases for the search. The first phase employs the common hyperheuristic framework with the operator selection mechanism. The aim of this phase is to produce a good enough candidate solution as soon as possible. After a certain period of time, the system must learn the best sequence of operators to be executed. We use several neighborhoods of the current solution as the low level operators: *generation operation*, *removal and add operator*, *cross operator*, *cut and add operator*, *greedy cut and add operator 1*, *greedy cut and add operator 2*, *greedy removal and add operator 1*, *greedy removal and add operator 2*, *two points cross operator*, *swap operator*. In the second phase, the goal is to achieve large ‘jump’ out of unpromising regions of the solution search space. A set of rules are defined to guide the construction of new solutions in order to broaden the horizon of the search process. Note that using these rules, the search is allowed to make some moves into infeasible regions in the hope that it quickly reaches a good solution region. The rules are established to destroy a part of the current solution and then repair it immediately or later. Once the sequence of rules has been applied, the solutions return to the first phase to continue the search.

4 Experiments

The experiments for bus driver scheduling problem discussed in this paper contain tasks (start time and finish time) randomly selected from a real problem. The details of the instances can be found in www.lac.inpe.br/~lorena/instancias.html.

All the tests are performed with a Intel Core2(TM) 2 Duo CPU (2 GHz, 2Gb of RAM). The algorithm was developed in the Java language. The best known schedules were obtained by [3] with a Intel Celeron® processor of 2.0 GHz and 256Mb of RAM memory.

Number of tasks	Best known solutions			Number of drivers	Total cost	Deviation to best known (%)	Times (seconds)
	Number of drivers	Total cost	Times (seconds)				
25	12	2371	0.10	12	2341	-1.26	0.46
50	20	2600	4.63	20	2600	0	0.50
100	40	7395	1.88	40	7395	0	0.68
250	81	8854	70.90	81	8632	-2.50	19.16
500	145	9716	6567.80	145	9842	1.29	692.31
Average						-0.49	

Table 1 Comparative results against the best known schedules

The Table 1 presents our results. The first column gives the numbers of tasks of the instances. The three following columns respectively give the number of drivers, the total cost and the computation times for the best known solutions from the literature [3]. The next columns present the results obtained with our TPPH. In Table 1, the computational results show that for a same number of drivers obtained, the solutions present a total cost of 0.49% less comparing with the best known solutions. In addition, computation times are much faster for larger cases.

5 Conclusions

In this paper we have introduced a novel type of hyperheuristic for bus driver scheduling problem, and have studied its performance on a set of problems. The results indicate that the two phase parallel hyperheuristic (TPPH) can produce good quality solutions.

References

1. A. Wren, S. Fores, A. Kwan, R. Kwan, M. Parker and L. Proll, A flexible system for scheduling drivers, *Journal of Scheduling*, 6, 437–455 (2003).
2. Li J. and Kwan R.S.K, A Self-Adjusting Algorithm for Driver Scheduling, *Journal of Heuristics*, 11 (4), 351-367 (2005).
3. G. R. Mauri, L. A. N. Lorena: A new hybrid heuristic for driver scheduling, *International Journal of Hybrid Intelligent Systems*, 4 (1), 39-47(2007).
4. H. R. Lourenc_o, J. P. Paix~ao, and R. Portugal. Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation Science*, 35:331–343(2001).
5. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: *Handbook of Metaheuristics*, chap. 16: Hyper-Heuristics: An Emerging Direction in Modern Search Technology, 457–474. Springer, New York (2003)
6. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: An emerging direction in modern search technology. In: F. Glover, G. Kochenberger (eds.) *Handbook of Metaheuristics*, 457-474. Kluwer (2003)
7. Shaw, P, A new local search algorithm providing high quality solutions to vehicle routing problems, Working paper, University of Strathclyde, Glasgow, Scotland (1997).
8. Kwan, R. S. K, Bus and train driver scheduling. In J. Y.-T. Leung (ed.), *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Chapter 51, 1-20. (2004)
9. Richard S. Sutton, Andrew G. Barto , *Reinforcement Learning: An Introduction*. IEEE Transactions on Neural Networks 9(5), 1054-1054 (1998).
10. Chew, K.L., Pang, J., Liu, Q.Z., Ou, J.H., Teo(2001): C.P.: An optimization based approach to the train operator scheduling problem at Singapore MRT. *Ann. Oper. Res.* 108(1-4), 111–122
11. Meignan D.,Créput, J.-C.,and Koukam (2010): A Coalition-based metaheuristic: A self-adaptive metaheuristic using reinforcement learning and mimetism, *Journal of Heuristics* 16: 859–879

Off-shore Helicopter Routing with Asymmetric Demands and Refueling Options

Ward Romeijnders · Gerard Sierksma

This paper investigates the problem of transporting crew members to off-shore gas drilling platforms through helicopters. The demands are asymmetric, i.e., the number of crew members transported to a platform is not necessarily equal to the number of crew members picked up from that platform. The objective of the problem is to find a flight schedule of minimal total length such that all demands for crew exchanges are satisfied. The number of passengers may of course not exceed the helicopter capacity, and the flight length should not exceed its range. This problem is NP-hard, since of the



already NP-hard traveling salesman problem. We use a max-regret heuristic that finds feasible flight schedules within very reasonable time limits. Simulations, using random transportation demands, reveal that this heuristic outperforms all well known other heuristics. The heuristic considers platforms far away from the airport first, because

Ward Romeijnders
University of Groningen, 9700 AB Groningen, The Netherlands
E-mail: w.romeijnders@student.rug.nl

Gerard Sierksma
Department of Operations, University of Groningen, 9700 AB Groningen, The Netherlands
E-mail: g.sierksma@rug.nl

too many helicopter flights to one of these platforms can be considered as a too high 'regret'. The max-regret heuristic is a constructive heuristic that executes the clustering and routing processes simultaneously.

If a platform cannot be reached, because it is located too far away, then refueling along the route is needed. We investigate how many refueling installations should be build and where. We do not take into account the cost of the refueling installations, but merely investigate the effect on the transportation costs. Although this paper focuses on the situation of 51 Dutch production platforms in the North Sea, the results are valid for many other situations all over the world, because the location topology of an airport on the mainland and all platforms in a sector with the airport as vertex is pretty common.

Key words: vehicle routing; helicopter routing; split demand; max-regret heuristic; refueling

Scheduling Wafer Release in Clustered Photolithography Tools

Kyungsu Park • James R. Morrison

In semiconductor wafer fabricators (fabs) with high product mix, tool setups are common. In the 450 mm era, it is anticipated that there will be a significant increase in setups driven by a reduction in lot size. To address the throughput loss associated with setups in clustered photolithography tools, pre-scan wafer buffers have often been used to reduce the effect of track setups. To maximize the use of the pre-scan buffer and thereby maximize throughput, wafers are typically admitted and advanced in an opportunistic manner. That is, the wafers progress when the wafer, wafer handling robot and next process module are available. At the tool load lock, wafers are admitted as soon as possible with an opportunistic wafer admission policy. Since the scanner is typically the bottleneck of the cluster, the opportunistic advancement enables to pre-scan buffer to build a surplus of wafers that can be supplied to the scanner in the event of pre-scan track setup delays. This serves to minimize the loss of throughput at the scan process.

While the opportunistic wafer admission policy guarantees minimal throughput loss, open loop pre-scan buffer level management may cause the wafers to incur unnecessary waiting time, especially in the buffer. The increase in wafer residency time due to buffer use may result in higher contamination levels resulting in yield loss. In [1], they demonstrated a close relation between wafer waiting times at key production stages and manufacturing yield. Some studies have also assumed that the reduction of wafer residency time in semiconductor fabrication will result in yield improvement; c.f. [2-4].

In addition, opportunistic wafer admission causes deployment capacity loss as well. Lots are admitted into the tool earlier than needed and lose a chance to be deployed elsewhere.

Our goal is to develop a computationally efficient scheduling algorithm to determine when wafers should be admitted to a clustered photolithography tool to guarantee no loss of throughput while minimizing the wafer residency time. In [5], relatively computationally inefficient wafer admission control algorithms employing custom heuristics were developed to address the problem. In particular, effects of the wafer admission algorithms based on numerical experiments are studied in [5]. In this paper, we develop a significantly more efficient linear programming (LP) model as a method to solve the admission control problem.

As in [5], we ignore the wafer handling robots and abstract to a flow line model for the tool. It has been shown in [6] that such models can well serve to predict throughput and cycle time for clustered photolithography tools. This abstraction serves to simplify the scheduling

problem and dovetails naturally with a hierarchical approach to scheduling the tools. The result can be used as guidance of wafer admission timing. Let $X_{w,m}$ denote the entry time of wafer w into process m . Let C_w denote the exit time of wafer w from the tools. The flow line consists of M processes, each with $R(m)$ modules devoted to it. We use a_w and τ_m to denote the arrival time of wafer w and the deterministic service time for process m , respectively. The wafer progression in flow line models can be represented by the elementary evolution equations (EEEs) (see [6]):

$$\begin{aligned} X_{w,1} &= \max \{ a_w, X_{w-R(1),2} \}, \\ X_{w,i+1} &= \max \{ X_{w,i} + \tau_i, X_{w-R(i),i+1} \}, \\ X_{w,m} &= \max \{ X_{w,M-1} + \tau_{M-1}, X_{w-1,m} + \tau_M \}, \\ C_w &= X_{w,m} + \tau_M, \end{aligned}$$

for $w > 0$, $i = 1, \dots, m-1$, with initial conditions $X_i(w) = -\infty$, for $w \leq 0$ and $i = 1, \dots, m$.

Here we show that the problem of maximizing throughput and subsequently minimizing wafer residency time can be formulated as two successive LPs. This serves to express the problem introduced in [5] into standard form as well as enabling significant computational reduction.

Suppose that there are G lots of L wafers each. The arrival time of lot g is A_g . Without loss of generality, A_g is a positive value. Let $c(g)$ and τ_m^i denote the class of lot g and the process time of a wafer of class i at process m . For photolithography tools, a reticle alignment process is conducted for every first wafer of each lot at scan process S and it consumes τ_{align} units of time. If the lot class changes, the pre-scan track setup begins when all modules before and including process P are vacant. It takes τ_{setup} units of time to conduct the setup (during which time all pre-scan modules must be vacant). Let $I(g)$ be the indicator of whether a setup occurs prior to lot g processing. That is, when lot g and $(g-1)$ have different class, $I(g) = 1$; it is 0, otherwise. The detailed algorithms follow.

1st Linear Program (LP1): Given the anticipated lot arrival times and setup durations, we first calculate the earliest wafer completion times from the scanner under the opportunistic wafer admission policy. The LP model follows. The decision variables are $X_{w,m}$ and C_w .

$$\begin{aligned} &\text{Minimize } \sum_{w=L+1}^{(G+1)*L} C_w & (1) \\ &\text{Subject to} \\ &X_{w,m} = -N & \text{for } m=1, \dots, M \text{ and } w=1, \dots, L, & (2) \\ &X_{L*g+1,1} \geq I(g)*(A_g + \tau_{setup}) & \text{for } g=1, \dots, G, & (3) \\ &X_{L*g+1,1} \geq I(g)*(X_{L*g,P+1} + \tau_{setup}) & \text{for } g=1, \dots, G, & (4) \\ &X_{L*g+1,1} \geq (1-I(g))*A_g & \text{for } g=1, \dots, G, & (5) \\ &X_{L*g+1,1} \geq (1-I(g))*X_{L*g+1-R(1),2} & \text{for } g=1, \dots, G, & (6) \\ &X_{L*g+w,1} \geq A_g & \text{for } g=1, \dots, G \text{ and } w=2, \dots, L, & (7) \\ &X_{L*g+w,1} \geq X_{L*g+w-R(1),2} & \text{for } g=1, \dots, G \text{ and } w=2, \dots, L, & (8) \\ &X_{L*g+w,m} \geq X_{L*g+w,m-1} + \tau_{m-1}^{c(g)} & \text{for } m=2, \dots, M-1, g=1, \dots, G \text{ and } w=1, \dots, L, & (9) \\ &X_{L*g+w,m} \geq X_{L*g+w-R(m),m+1} & \text{for } m=2, \dots, M-1, g=1, \dots, G \text{ and } w=1, \dots, L, & (10) \\ &X_{L*g+1,S+1} \geq X_{L*g+1,S} + \tau_{S+1}^{c(g)} + \tau_{align} & \text{for } g=1, \dots, G, & (11) \\ &X_{L*g+w,m} \geq X_{L*g+w,M-1} + \tau_{M-1}^{c(g)} & \text{for } g=1, \dots, G \text{ and } w=1, \dots, L, & (12) \\ &X_{L*g+w,m} \geq X_{L*g+w-R(M),m} + \tau_m^{c(g)} & \text{for } g=1, \dots, G \text{ and } w=1, \dots, L, & (13) \\ &C_{L*g+w} \geq X_{L*g+w,m} + \tau_m^{c(g)} & \text{for } g=1, \dots, G \text{ and } w=1, \dots, L. & (14) \end{aligned}$$

Here, N is a large positive number. Note that we put one additional lot before the regular lots for initialization; equation (2) is for the initialization. We use C'_w to denote an optimal solution for C_w from LP1.

2nd Linear Program (LP2): Calculate the latest admission time for each wafer that will still guarantee that the wafer will exit the scanner as early as possible. The LP model follows. The decision variables are $X_{w,m}$.

$$\begin{aligned} &\text{Maximize } \sum_{g=1}^G \sum_{w=1}^L X_{g*L+w,1} & (15) \\ &\text{Subject to equations (2),(3),...,(14), and} \end{aligned}$$

$$C_{L*g+w} = C'_{L*g+w}, \quad \text{for } g=1, \dots, G \text{ and } w=1, \dots, L. \quad (16)$$

LP1 ensures maximum throughput while LP2 minimizes the residency time of each wafer in the tool. Equations (1) and (16) guarantee maximum throughput. Equation (15) minimizes the residency time subject to the maximum throughput. One caveat is that we assume knowledge of lot arrival times and setup durations. Randomness in such times can readily be allowed by using worst case or inflated times for safety.

We conducted simulation studies on a clustered photolithography tool model to determine the performance of the LP approach. The models are based on [6] and have a 24 wafer pre-scan buffer. Setups on the track occur when changing lot type and the 1st wafer of each lot prompts delay for a reticle (re)alignment at the scanner. There are three classes of lots. The average number of lots of the same class processed sequentially is given by train size $T = 3$ lots. Table 1 provides computation time as a function of the number of lots, for four methods: Custom algorithm (CA) of [5] in MATLAB, LP method in MATLAB, combination of CA and LP method in MATLAB, LP method via ILOG's Optimization Programming Language (OPL). The unit of time is seconds. CA uses a kind of complete enumeration approach, which tries and evaluates all the possible entry time for integer process times and arrival times. Combination of CA and LP method means solve LP1 by CA in MATLAB and LP2 by LP via OPL. We simulate on an Intel Core 2 Quad CPU at 2.40 GHz with 2.00 GB of RAM.

number of lot		1	2	3	4	5	6	7	8	9	10
MATLAB	Trial method	0.25	0.69	2.14	3.69	6.63	7.60	10.46	21.45	21.67	27.56
	LP	1.47	9.41	27.26	60.23	109.37	179.30	271.69	384.11	538.81	x
	Trial+LP2	1.24	7.92	22.39	49.42	101.45	176.41	267.80	377.13	502.07	722.41
ILOG's OPL	LP	2.02	2.17	2.06	2.11	2.19	2.18	2.21	2.27	2.22	2.27
number of lot		15	20	25	30	35	40	45	50	1800	
MATLAB	Trial method	21.76	75.41	146.68	221.36	281.57	318.18	487.43	1278.78	54439.00	
	LP	x	x	x	x	x	x	x	x	x	
	Trial+LP2	x	x	x	x	x	x	x	x	x	
ILOG's OPL	LP	2.43	2.44	3.02	3.41	4.54	5.61	6.07	x	x	

Table 1. Computation time comparison.

Though LP method is slower than CA method in MATLAB, LP can be solved with very small computation time by ILOG's OPL. However, LP models fail for a large number of lots (more than 45 lots). On the other hand, the CA method allows a larger number of lots even though it can have exceptionally heavy computation time if there is long successive sequence of the same class of lot. Simulation variation causes non-monotonicity.

In this paper we strive to maximize tool throughput and reduce wafer residency time using an LP model. The proposed wafer admission control may improve deployment loss and yield. The simulation results show that the time comparison between several methods.

Acknowledgements This work was supported in part by National Research Foundation of Korea (NRF) Grant N01100263.

References

1. Wen-Chi Chang, Max Yu, Ray Wu, Claire Chen, Joey Chen, C. Y. Hsieh, and C. K. Wang, "Yield improvement through cycle time and process fluctuation analyses," Proceedings of the 2001 IEEE ISMS, pp. 267-270 (2001)
2. Krishna Srinivasan, Raka Sandell, and Steven Brown, "Correlation between yield and waiting time: A quantitative study," IEEE/CPMT IEMTS, pp. 65-69 (1995)
3. Lawrence M. Wein, "On the relationship between yield and cycle time in semiconductor wafer fabrication," IEEE TSM, Vol. 5, No.2, pp. 156-158 (1992)
4. Doron Meyersdorf and Taho Yang, "Cycle time reduction for semiconductor wafer fabrication facilities," IEEE/SEMI ASMC, pp. 418-423 (1997)
5. Kyungsu Park and James R. Morrison, "Wafer admission control for clustered photolithography tools," Proceedings of the 2010 ASMC, pp. 220-225 (2010)
6. James R. Morrison, "Deterministic flow lines with applications," IEEE TASE, Vol. 7, No. 2, pp. 228-239 (2010)

Algorithmic Parallelization in Multi-Criteria Semiconductor Scheduling

Chase E. Rainwater • Scott J. Mason • John W. Fowler • Lars Mönch

Scheduling is a decision making process involving the allocation of (often scarce) resources to tasks over some time period in order to optimize one or more objective functions or performance measures of interest [8]. Many practically-motivated scheduling problems are found in both manufacturing and service industries. While effective scheduling can promote increased efficiencies and profits for a company, ineffective scheduling can cause high cycle times, customer dissatisfaction and in the extreme, lost sales and customer goodwill. Consider the highly capital intensive world of semiconductor manufacturing, arguably the most complex manufacturing process in existence today. Poor production scheduling can cause a manufacturer to ineffectively utilize installed capacity and/or can cause a manufacturer to "miss the market" opportunity for selling their product at a premium price, given the short product life cycles associated with "high tech" products. Further, customers may choose to take their business elsewhere if the manufacturing facility with which they have contracted is unable to deliver their products in a timely fashion.

Unfortunately, most semiconductor manufacturers do not truly schedule their factories' equipment sets (tools), but rather make local dispatching decisions. This reality is necessitated by the fact that most scheduling problems of practical interest/importance, even for a single performance measure of interest, are NP-hard or strongly NP-hard. Therefore, dispatching approaches have become the state of the practice. These approaches provide fast, hopefully "good" solution options for fab operators and management to meet desired performance targets, albeit in a localized, myopic, time horizon-less manner. While this may result in apparently "good" local decisions, dispatching decisions generally do not account for upstream, downstream, and/or factory-wide ramifications. They also do not comprehend the multiple, usually competing, performance measures (see [10] for a discussion of multiple criteria scheduling).

Recent advances in parallel computing technologies, such as University of Arkansas's Red Diamond Cluster, Clemson's Palmetto Cluster, and Arizona State's Saguaro Cluster, have afforded a wide array of researchers with up-to-now unimaginable computational horsepower for analyzing challenging research questions. Intel Corporation, the world's largest manufacturer of integrated circuits, has pioneered many of these advances with their dual- and quad-core integrated circuits (ICs) that contain two and four central processing units (CPUs) on a single chip. These massive academic parallel computing clusters take these quad-core

Chase E. Rainwater
University of Arkansas
E-mail: cer@uark.edu

Scott J. Mason
Clemson University
E-mail: mason@clemson.edu

John W. Fowler
Arizona State University
E-mail: john.fowler@asu.edu

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

chips and put them together in a single computing node, thereby creating an eight-core "machine," of which now there are over 9,000 operating in parallel on Clemson's Palmetto Cluster, for example. The availability of such massive amounts of computational horsepower and the importance of semiconductor scheduling motivates our collaborative research project

Our overarching research agenda is to exploit the power of parallel computing technologies to develop new solution methods and algorithms for analyzing challenging, multi-level, multi-criteria scheduling problems of practical and economic importance. Our focus will be on both factory- and work center/toolset level scheduling decisions. The emergence of the effective class of so-called parallel meta-heuristics [9] suggests an opportunity to generate improved solutions on far larger scheduling problems than ever attempted. Specifically, many factory scheduling problems have a decomposable structure that can be exploited to improve parallel solution approaches. For example, well-known job scheduling methodologies, such as the Shifting Bottleneck Heuristic, lend themselves nicely to parallelization because the heuristic decomposes the overall scheduling problem into a series of smaller scheduling problems (subproblems) for single tools or tool groups. In this approach, the natural master/subproblem structure provides an opportunity for so-called domain parallelism to evaluate the partitioned regions of the search space.

An architecture for a parallel implementation of the shifting bottleneck heuristic (SBH) for complex job shops like wafer fabs is proposed in [2] using a cluster computer that consists of a master node and several compute nodes. It is demonstrated by applying the shifting bottleneck heuristic in a rolling horizon setting using discrete-event simulation that the parallel version of the shifting bottleneck heuristic is faster than the sequential implementation when the amount of computing time for the resulting subproblems is large enough. When we assume that we can use a single compute node for each M tool group subproblem and that all subproblems require

approximately the same computing time, then we can expect a speedup of about $\frac{M+1}{2}$ when communication costs are minimal. Therefore, we are able to apply more sophisticated subproblem solution procedures within the shifting bottleneck heuristic. Now it is also possible to assess the performance of the SBH by conducting simulation studies for a much larger simulation horizon.

A more coarse-grained decomposition strategy for complex job shops is proposed in [5]. Here, the overall scheduling problem is decomposed using an upper planning layer that determines expected start and completion dates for certain aggregated operations of the jobs. As a result, we obtain a set of smaller job shop scheduling problems. Each of them is solved using the shifting bottleneck heuristic. In a second step, modified start dates and completion dates iteratively are exchanged among the smaller scheduling problems. These scheduling problems with modified data are solved again to improve the performance measures. This approach is distributed among several computers within the multi-agent-system FABMAS [6].

Unfortunately, domain parallelism typically implemented in the meta-heuristic realm suffers from the lack of synchronization opportunities, which ultimately results in unwanted time delays. However, we believe that our recent research efforts in multiple orders per job (moj) scheduling lend themselves favorably to methodologies that evaluate improved synchronization-based approaches made possible by manageable message passing interface (MPI) implementations (e.g. Open MPI).

Our previous scheduling and multi-criteria scheduling efforts will serve to galvanize our proposed research efforts towards a successful outcome. We have examined scheduling problems involving single and parallel machines with most of the challenging complexities contained in semiconductor manufacturing including recirculating/re-entrant flow, auxiliary resource requirements (e.g., reticles for photolithography steppers), batch processing (e.g., diffusion ovens), and sequence-dependent setups (e.g. ion implanters). In addition, we have investigated scheduling under multiple criteria for parallel machines [1], a multi-criteria shifting bottleneck heuristic using a desirability function-based approach [7], and we developed the integrated preference function methodology [3] that allows one to compare sets of near-Pareto optimal solutions with one another. In this presentation, we discuss our efforts

in investigating single- and parallel-machine multiple orders per job scheduling problems that contain a multi-level structure (i.e., order-to-job assignment and job scheduling) to evaluate potential benefits from parallel computing architectures. The moj scheduling problems specified in our research are motivated by 300-mm semiconductor manufacturing where customer orders are produced on less than a full lot of wafers, thereby necessitating the grouping of multiple orders into the same transport container (Front Opening Unified Pod). These comments are even more important in the presence of multiple objective functions. We will discuss our experience with developing strategies for decomposing this class of problems to best exploit parallel architectures. This discussion will investigate the structure of resulting subproblems as well as the significance of being able to solve subproblems efficiently using serial algorithms. Further, particular emphasis will be given to ways that synchronization opportunities can improve computational efficiencies. We will conclude with benchmark results for parallelized moj solution procedures versus state-of-the-art serial algorithms currently available in the literature (e.g. [4]).

In the future, our research effort focus will be classical scheduling problems under multiple criteria and on cluster tools. Cluster tools are multi-chamber tools characterized by a central control chamber containing one or more wafer handling robots, parallel load locks (i.e., load ports where up to four lots can be put onto the tool at the same time for processing), and a number of plug-and-play processing chambers that can be arranged for a variety of lot processing configurations (e.g., deposition processes, etch processes, etc.). We will collaborate with key global experts in semiconductor manufacturing scheduling and parallel computing, that will help us to develop the most effective solution methodologies under a parallel computation paradigm.

References

1. Cochran, J.K., Horng, S-M, and Fowler, J.W., "A Multi-Population Genetic Algorithm to Solve Multi-Objective Scheduling Problems for Parallel Machines", *Computers and Operations Research*, 30(7), 1087-1102, (2003).
2. Driessel, R., Hömig, U., Mönch, L., Schiffmann, W., A Parallel Shifting Bottleneck Heuristic for Scheduling Complex Job Shops: Architecture and Performance Assessment. Proceedings IEEE Conference on Automation Science and Engineering (CASE) 2010, 81-86, (2010).
3. Fowler, J.W., Kim, B., Carlyle, W.M., Gel, E.S., and Horng, S.-M., "Evaluating Solution Sets of *a Posteriori* Solution Techniques for Bi-Criteria Combinatorial Optimization Problems", *Journal of Scheduling*, Vol. 8, No. 1, pp. 75-96, 2005.
4. Laub, J.D., Fowler, J.W., and Keha, A.B., "Minimizing Makespan with Multiple-orders-per-job in a Two-machine Flowshop", *European Journal of Operations Research*, 182(1), 64-79, (2007).
5. Mönch, L., Drießel, R., A Distributed Shifting Bottleneck Heuristic for Complex Job Shops. *Computers & Industrial Engineering*, (49), 673-680, (2005).
6. Mönch, L., Stehli, M., Zimmermann, J., Habenicht, I., The FABMAS Multi-Agent-System Prototype for Production Control of Waferfabs: Design, Implementation, and Performance Assessment. *Production Planning & Control*, 17(7), 701-716, (2006).
7. Pfund, M.E., Balasubramanian, H., Fowler, J.W., Mason, S.J., and Rose, O., "A Multi-criteria Approach for Scheduling Semiconductor Wafer Fabrication Facilities", *Journal of Scheduling*, 11(1), 29-47, (2008).
8. Pinedo, M., *Scheduling: Theory, Algorithms, and Systems*, second ed., Prentice Hall, (2001).
9. Talbi, E.-G. (ed.), *Parallel Combinatorial Optimization*, Wiley, Hoboken, New Jersey, (2006).
10. T'Kindt, V., Billaut, J., *Multicriteria Scheduling*, first ed. Springer, (2002).

Scheduling job families on parallel machines with a bi-criteria objective function under time constraints

Ali Obeid · Stéphane Dauzère-Pérès ·
Claude Yugma

1 Introduction

Semiconductor Manufacturing is getting more and more competitive and industries are looking for strategies to improve productivity, decrease cost and enhance product quality. Advanced Scheduling and Advanced Process Control (APC) systems support these objectives. Scheduling means assigning and sequencing jobs on machines to minimize some given objectives under a set of constraints. Run-To-Run (R2R) control is widely used to enhance the quality of products by compensating for process drifts and adjusting machine parameters. Collecting data at both machine and process levels helps in the detection of current process drifts and/or machine degradation, as well as in the prediction of possible faults. Scheduling and control can be considered as mutually related issues in Semiconductor Manufacturing.

In this abstract, we address the scheduling of lots (jobs) of different product types (job families) on parallel machines, where not all machines are able (qualified) to process all job families (non-identical machines). A special time constraint, associated to each job family, should be satisfied for a machine to remain qualified for processing a job family. This constraint imposes that there must be at most a given time threshold between the processing of two jobs of the family on the machine. Moreover, we consider that a setup time is required before starting the first job of a family. This problem was addressed in [5]. It comes from Semiconductor Manufacturing, where the problem of scheduling lots in the lithography area can be seen as a scheduling problem on parallel machines with job family setups (also called s-batching) [4]. Time constraints are relevant when Advanced Process Control constraints are considered in such scheduling problems.

A short literature review is performed in the next section. To approach this problem, we developed and tested several heuristics which are described in Section 3. A sample of the numerical experiments is provided in Section 4.

Ali Obeid, Stéphane Dauzère-Pérès, Claude Yugma
École des Mines de Saint-Étienne
880, route de Mimet, 13541 Gardanne, France
E-mail: {obeid, dauzere-peres, yugma}@emse.fr

2 Short literature review

There are very few articles which deal with scheduling decisions while integrating APC constraints. The impact of APC on scheduling performances is analyzed by [3]. They also study the scheduling of job families on parallel machines. However, they consider that machines are identical, that qualification runs can be scheduled and that the threshold between two jobs of the same family is given in terms of number of jobs. We consider non-identical parallel machines and assume that qualification runs cannot be scheduled and will be performed after the scheduling horizon. The problem becomes more complicated, since the assignment of jobs to machines is critical to avoid qualification runs. Finally, we consider a threshold expressed in terms of time instead of number of jobs. Both threshold types are actually relevant and are related. [6] study the interaction between scheduling and APC on one machine, with setup times between two job families, and a qualification run when the Run-To-Run (R2R) constraint is not satisfied. They show that the Makespan problem on a single machine with multiple job families is NP-hard. Another example of integration of APC constraints in scheduling decisions can be found in [1] and [2], where measurement operations are optimally scheduled to minimize the risk of losing products in jobs. Hence, we address a new scheduling problem in which there is a time constraint on jobs of the same family, i.e. the time interval between two consecutive jobs of the same family should be smaller than a given threshold. As mentioned above, this time constraint is inspired from the needs of APC systems, and in particular R2R control loops for a given product type on a machine, that require to regularly collect data for the qualified product types on machines. In the following section, we review the heuristics we propose to solve our problem.

3 Heuristics

3.1 List heuristics

The jobs of different families on the parallel machines are scheduled by using two priority rules: Earliest Time Threshold (ETT) and Shortest Processing Time (SPT). The Earliest Time Threshold rule first schedules jobs of the family with the earliest threshold, i.e. the most urgent family first to keep the machine qualified.

3.2 Recursive heuristic

The general idea of this heuristic is to schedule jobs by accepting each time one qualification loss or more. More precisely, we consider the solution obtained by the list heuristic, and reapply the heuristic after changing the initial qualification scheme. The perturbations in the qualification scheme are chosen from the set of disqualified machines in the solution obtained by the list heuristic. In other words, after scheduling the jobs using the list heuristic, we examine the resulting solution to verify whether the machines are still capable (qualified) to execute the jobs, i.e. whether thresholds are satisfied or not. If this is not the case, we change the data to accept some threshold violations, and reapply the list heuristic recursively to improve the solution.

3.3 Scheduling-centric heuristic

The main goal of the scheduling-centric heuristic is to minimize setup times. Recall that a setup time is necessary when two jobs of different families are scheduled consecutively on a machine. The heuristic starts by placing first the jobs of the family with the shortest threshold. When adding a job implies that the threshold of another family will be violated on the machine, the heuristic shifts to a job of this other family.

3.4 Qualification-centric heuristic

The main objective of the qualification-centric heuristic is to minimize the total number of violations of the time constraint of each family on the machines on which the family is qualified. To do this, the qualified family with still a job to schedule and the shortest remaining time threshold is selected first on a machine. A job in this family is scheduled on the machine, and its completion time is updated. The remaining time threshold of the family is then reset on the machine. If it is no longer possible to schedule a job of a family before its remaining time threshold, then the machine is disqualified for the family.

After scheduling all the available jobs on the qualified machines while focusing on maintaining machine qualifications, we then try to reduce the impact of setup times between job families by trying to schedule jobs of the same family consecutively. To do this, we consider the last job of the sequence on a machine, and try to schedule this job as early as possible with a job of the same family. We then check whether the machine qualifications are still valid in the resulting schedule. If this is the case, then the change is accepted, otherwise the new schedule is rejected. The process is repeated for all machines and for each last job until no change is possible.

This heuristic is still being tested and its results will be presented at the conference.

4 Computational experiments

Table 1 Heuristics applied on test instances - Sum of completion times (best solution in bold).

<i>Instance</i>	Sum of completion times			
	List Heuristic	Recursive Heuristic	Sched-centric Heuristic	Exact Solution
1	2322	1617	1617	1617
2	758	758	578	578
3	1644	1698	1423	1275
4	1992	1992	1992	-
5	1547	1547	1136	1129
6	1161	1068	1048	873
7	1358	1316	1316	1316
8	912	874	618	574
9	1109	1069	1032	961
10	4881	4881	4303	-

Table 1 and 2 present some results on various test instances. The results show that the scheduling-centric heuristic gives better solutions in terms of the sum of completion times and, for most observations, in terms of number of machine qualifications. The experiments were performed by considering one criterion at a time. Detailed results for each heuristic as well as a comparison of the results between the list, recursive, scheduling-centric and qualification-centric heuristics will be presented at the conference. The results of a multi-criteria analysis will be shown and discussed as well.

Table 2 Heuristics applied on test instances - Machine qualifications (best solution in bold).

Instance	Sum of qualification losses (threshold violations)			
	List Heuristic	Recursive Heuristic	Sched-centric Heuristic	Exact Solution
1	2	1	1	1
2	1	1	0	0
3	4	4	4	0
4	2	2	1	0
5	0	0	0	0
6	6	2	2	1
7	2	1	1	1
8	5	4	1	1
9	5	5	5	1
10	1	1	1	-

References

1. B. Detienne, S. Dauzère-Pérès, and C. Yugma, Scheduling Inspection Operations Subject to a Fixed Production Schedule, *4th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2009)*, 581-593 (2009)
2. B. Detienne, S. Dauzère-Pérès, and C. Yugma, Scheduling jobs on parallel machines to minimize a regular step total cost function, to appear in *Journal of Scheduling* (2011)
3. L. Li, and F. Qiao, The impact of the qual-run requirements of APC on the scheduling performance in semiconductor manufacturing, *4th IEEE Conference on Automation Science and Engineering*, 242-246 (2008)
4. L. Möñch, J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose, Scheduling Semiconductor Manufacturing operations: Problems, Solution Techniques, and Future Challenges, to appear in *Journal of Scheduling* (2011)
5. A. Obeid, S. Dauzère-Pérès, C. Yugma, and A. Ferreira, Scheduling Job Families on Non-Identical Parallel Machines under Run-To-Run Control Constraints, 12th International Workshop on Project Management and Scheduling, 309-313 (2010)
6. C. Yiwei, E. Kutanoglu, J. Hasenbein, and J. Qin, Scheduling with Advanced Process Control Constraints, www.me.utexas.edu/~has/apc09.pdf (2009)

Rapid Development of University Course Timetables

Hana Rudová · Tomáš Müller

1 Introduction

Many universities continue to solve timetabling problems manually even though there are a variety of timetabling systems¹ applicable to these problems [3,1]. The often tedious process of initially setting up an automated system and making adjustments for the specific needs of an institution is one of the critical reasons for this. We will discuss an application of the comprehensive timetabling system UniTime²[4] at a large college with 10,800 students. Here the generated timetables were published eight weeks after our first meeting with the schedule manager. Our goal is to describe how such a rapid implementation of an automated university course timetabling system can be achieved even for large problems.

2 Timetabling Process

The course timetabling problem for the Faculty of Arts at Masaryk University had been solved manually until the Fall 2010 semester. The initial process consisted of manual creation of departmental timetables by about 40 schedule deputies. These timetables, containing all time assignments and most room assignments for the department classes, provided the initial input for the central schedule manager. Unfortunately, even these partial timetables were inconsistent. They did not reflect the availability of shared resources (rooms) and substantial changes to the proposed assignments were necessary. The amount of changes required was estimated at 30-40% by the central schedule manager who was responsible for creation of the complete timetable. Partial timetables

Hana Rudová
Faculty of Informatics, Masaryk University
Botanická 68a, Brno 602 00, Czech Republic
E-mail: hanka@fi.muni.cz

Tomáš Müller
Space Management and Academic Scheduling, Purdue University
400 Centennial Mall Drive, West Lafayette, IN 47907-2016, USA
E-mail: muller@unitime.org

¹ <http://www.asap.cs.nott.ac.uk/watt/applications>

² <http://www.unitime.org>

from each of the schedule deputies were submitted via MS Excel files containing tables with assignments. These tables were manually modified during the process. Finally a text form of the timetable was generated from Excel and imported into the Information System of Masaryk University (IS MU)³.

Since the college has been renovating some of its facilities, it was necessary to create timetables for Spring 2011 such that the number of available classrooms decreased from 65 to 49. Such a substantial change was one of the main reasons for our attempt to generate timetables automatically even within such a limited time period. Since training 44 schedule deputies in the use of UniTime was not manageable within the required time frame, we decided to use their manually created timetables as the primary input for automated generation of the complete college timetable. The remaining input data were retrieved from the IS MU. The overall initial input⁴ to the timetabling consists of the following data

- partial timetables created by individual schedule deputies: time assignments for all 1917 classes, partial room assignments (69 % of classes), class identifier (course identifier + optional section number), preferred room equipment (standard room, multi-media lab, computer lab), instructor name
- 49 rooms: identifier, building, capacity, equipment
- 584 instructors: identifier, name
- 1570 courses: identifier, name
- last-like semester enrollment to courses: course identifier, student (70689 records).

It is important to realize that each course may consist of several classes, i.e., one or more lectures and/or one or more seminars (note above that there are 1570 courses and 1917 classes). In most cases we identified the course structure automatically. The correct structure of the remaining courses was entered through the UniTime interface. Another important manual process was identifying courses having two or more identifiers (so called cross-listed courses). Many of these were identified automatically based on the assignment of multiple courses to the same time and room within a departmental timetable. The correct course structure is important to be able to schedule students meaningfully among lectures and seminars and to share resources among cross-listed courses if necessary [4]. After cross-listed courses and classes were identified, the number of courses and classes decreased to 1421 courses and 1746 classes. This means that there were 1746 entities requiring time and room assignments.

The goals of the timetabling problem in this instance are to assign times and rooms to all classes such that consistency of rooms and instructors is maintained, the number of student conflicts between classes are minimized, preferences on the room equipment are considered, and the assignments in the partial timetables are retained as much as possible. In addition, placement of classes to very early and very late times is discouraged—the college allowed extension of the time patterns from 7:30 am to 8:45 pm due to missing classrooms, but their use is certainly not perceived positively. Some other constraints were specified by the central schedule manager using the UniTime interface but the quantity of these was not significant.

The information contained in the partial manually created timetables was not complete. These timetables did not contain room assignment for 31 % of classes (often due to missing classrooms). Moreover, it was not possible to assign more than 48 %

³ <http://is.muni.cz>

⁴ After the first publication of the timetable, 7 rooms and 13 classes were added.

Table 1 Comparison of generated timetables.

Solution	Fully Automated	First Published	Finalized
Student conflicts	812	871	1,119
Time preferences (%)	92.34	92.53	89.20
Room preferences (%)	82.99	83.38	74.65
Selected time kept (%)	89.8	89.9	87.66
Selected room kept (%)	62.9	65.6	64.05
Broken hard constraints	0	10	71
Interactive time changes (%)	-	1.4	10.85
Interactive room changes (%)	-	6.7	20.95

of classes due to inconsistencies among hard constraints. The amount of preference data for creation of acceptable timetables was also very limited. At Purdue University, where UniTime is used for course timetabling [4], input data contain information about preferred times and rooms for each class. Here we only have information about one acceptable time and, for 69 % classes, about one acceptable room. In addition, we were able to ask for preferred room equipment for each class. For rooms, it was possible to add information about preferred building based on the building assigned in the partial timetables. The original room in a partial timetable was treated as a strongly preferred room. We also needed additional input for time preferences. Our decision was to assign a strong preference for the selected time in the partial timetable. A weaker preference was assigned to all time slots during the week falling one period before and one period after the selected time. This is very important since it gives some advice to the system when the original selected time cannot be used. Since there are 16 time slots each day, starting at 7:30 am and ending at 8:45 pm, this makes it possible to distinguish between instructors demanding early and late hours and choose a closely matching time at least. In addition, we generally discouraged placement to very early and late times.

To conclude, the strong preference for selected times and rooms allowed keeping the partial timetables as much as possible. Other added preferences were generated to create an acceptable solution for both time and room. The standard criteria for minimizing student conflicts maintained quality with respect to student programs.

3 Evaluation and Conclusion

The generated timetables are made available in the IS MU⁵. We would like to compare the quality of three following timetables: the first timetable generated automatically, the timetable published after the first set of interactive changes [2], and the finalized timetable. After the first publication of the timetable, four rooms were added to the available resources. This certainly lead to significant changes which are also clear when exploring the number of interactive changes in the finalized timetable.

Results are presented for all three timetables in Table 1. These include information about the number of student conflicts between classes and the percentage of satisfied time and room preferences related to standard optimization criteria. In addition, we

⁵ <http://is.muni.cz/rozvrh/?obdobi=5026&fakulta=1421&lang=en>

present information about the percentage of time and room assignments we were able to retain from the manually created timetables. The number of broken hard constraints during interactive timetabling, when the user can allow such inconsistencies (e.g., use too small rooms or rooms without required equipment), are also presented⁶. The last two lines of the table show the percentage of interactive changes that were made to the first published and the finalized timetable in comparison to the fully automated timetable.

Generated solutions were very well accepted by the central schedule manager. Even given the reduced number of rooms, it allowed high satisfaction of all requested criteria. The current semi-automated approach allows much better consideration of student conflicts, which were originally considered only on the level of partial departmental timetables. Importantly, the central schedule manager evaluated the decrease in her workload to be 30 %, which is a significant improvement especially given this was the first use of the system. Certainly the quality of the final automated timetable depends greatly on the quality of manually generated partial timetables. Our experience also indicates that a fully automated solution may not be acceptable, even in the future, due to administrative and political reasons within the Faculty of Arts. In the following semesters, we would like to allow departmental schedule deputies to enter and modify their input data directly in the application. However, this may be a slow transition as the acceptance of a complex automated system may not be easy.

We would like to point out that no changes were necessary to the constraint solver of the UniTime system and only a couple of minor changes were necessary in its interface to make the data input easier. The only extensions consisted of the preparation of scripts to load the data in a proper format and to minimize manual input by automatically generating some constraints. This automated constraint generation allowed detection of the course structure, preparation of initial preferences, adjustments to the desired room capacity for classes, etc.

In the future, we plan to continue with further application of the system at Masaryk University. We have already started to work on the course timetabling problem for the Faculty of Education, where there is an intent to have the system used by all departmental schedule deputies right from the start.

Acknowledgements We would like to thank to Vladimíra Maroušková who worked with us as the central schedule manager. Our thanks go to the team of the Information System at Masaryk University who helped us to get necessary data. Hana Rudová is supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research intent No. 0021622419.

References

1. Barry McCollum. University timetabling: Bridging the gap. In Edmund K. Burke and Hana Rudová, editors, *PATAT 2006*, pages 15–35. Masaryk University, 2006.
2. Tomáš Müller, Hana Rudová, and Keith Murray. Interactive course timetabling. In *MISTA 2009*, pages 732–736, Dublin, Ireland, 2009.
3. Zeb Nash. UK higher education: sector wide study of timetabling and resource scheduling: presentation of findings and discussion. In *PATAT 2010*. Queen's University Belfast, 2010.
4. Hana Rudová, Tomáš Müller, and Keith Murray. Complex university course timetabling. *Journal of Scheduling*, 2011. To appear. DOI 10.1007/s10951-010-0171-3.

⁶ For the finalized timetable, this number includes 17 unassigned classes. Their assignment was not included by the schedule manager due to complex modeling of the course structure.

Adapting Particle Swarm Optimization for the Flexible Flow Line

Mary E. Kurz • Vinodh Sankaran

1 Introduction

The flexible flow line scheduling problem is a branch of production scheduling and is found in industries such as printed circuit board and automobile manufacturing. It is well known that this problem is NP-hard [4]. We approach the problem by implementing a particle swarm optimization (PSO) with a random keys solution encoding. We propose two novel updating strategies, called “bounceback” and “wrap-around” to ensure the particles remain in the feasible region and evaluate the impacts of these strategies. The results indicate that the encoding and updates interact to impact solution quality.

2 Problem Formulation

A flexible flow line (FFL) consists of several stages in series where each stage consists of multiple parallel identical machines where at least one stage has more than one machine. A job should not revisit a stage which it already visited but jobs can skip stages. This scenario is found in manufacturing industries where a job does not require all operations, such as a printed circuit board manufacturing line. In addition to standard flowline assumptions, we have non-anticipatory sequence-dependent setup times between jobs at each stage. The objective in this paper is to minimize makespan. We can succinctly describe this problem as $FFL|s_{ij}|C_{max}$.

3 Particle swarm optimization

PSO is a population-based search algorithm developed by Kennedy and Eberhart [2]. Each particle “flies” with a velocity which can be adjusted by flying experience. An advantage of PSO over other metaheuristic approaches is the simplicity in structure. A swarm is composed of a set of S particles, each of which is moving in a D -dimensional space, composed of four pieces of information: *Position*: y_{ij}^k for particle i along dimension j in iteration k ; *Velocity*: v_{ij}^k for particle i along dimension j in iteration k ; *Objective function value*; *Personal best position*:

Mary E. Kurz

Department of Industrial Engineering, Clemson University

E-mail: mkurz@clemson.edu

Vinodh Sankaran

AT&T

E-mail: vs1096@att.com

p_{ij} for particle i along dimension j . In addition, a swarm must know the position of the particle that achieved the best-ever objective function value: the position is p_{gj} along dimension j .

Particles modify their position on a coordinate-by-coordinate basis by adding the current position to the current velocity, as shown in Equation (1).

$$y_{ij}^{k+1} = y_{ij}^k + v_{ij}^k \quad \forall i, j, k \quad (1)$$

Each element of the velocity vector is updated using a weighted combination of three factors: the current velocity; the difference between the current position and the particle's personal best position; and the difference between the current position and the swarm's global best position. The weight on the difference between the current position and the particle's personal best position is composed of a random number r_1 , between (0, 1), and a tunable learning factor, c_1 . The weight on the difference between the current position and the swarm's global best position is composed of a random number r_2 , between (0, 1), and a tunable learning factor, c_2 . Equation (2) illustrates the velocity updating equation, in which w is the weight on the current velocity. Eberhart and Shi [1] introduced the weighting parameter w to the velocity equation to balance local and global search.

$$v_{ij}^k = wv_{ij}^k + c_1r_1(p_{ij} - y_{ij}^k) + c_2r_2(p_{gj} - y_{ij}^k) \quad \forall i, j, k \quad (2)$$

A generic description of the PSO algorithm is as follows:

- Step 0: Initialize each particle position and velocity randomly. Go to Step 3.
- Step 1: Update each particle's velocity and position
- Step 2: Evaluate the objective function for each particle
- Step 3: Update the personal best particle for each particle and the global best particle
- Step 4: If the stopping conditions are met, return the global best particle. Else, go to Step 1.

4 Random keys representation for particle swarm representation

PSO operates in $\mathbb{R}^D$ and so cannot be directly applied to a permutation representation. We represent solutions for the flexible flow shop with random keys, following Kurz and Askin [4]. Each solution is represented by a particle with D equal to the number of jobs in the problem instance. For example, the particle $Y_1^k = (1.32, 1.22, 0.55, 0.35)$ is for a problem with four jobs and at least two machines at stage 1. The particle positions, $y_{ij}^k \in [0, M)$ where M is the number of machines in stage 1, serve as machine assignment and sort keys to decode the solution. The integer part is the machine to which the job is assigned and the fractional part serves as the sort key to sort the jobs assigned to each machine. The makespan is computed using these sequences on stage 1 and placing each job at each stage on the machine that allows it to complete processing earliest. In general, a lower fractional value for y_{ij}^k will mean that job j will be earlier on its machine while a larger value means it will be later on its machine.

In utilizing random keys for FFL in PSO, we are limited in the locations that the particles can occupy. We consider two alternatives to adapt the position update mechanism:

1. Bounceback: if $y_{ij}^k \geq M$ then $y_{ij}^k = 2M - y_{ij}^k$ or if $y_{ij}^k < 0$ then $y_{ij}^k = -y_{ij}^k$.
2. Wrap-around: if $y_{ij}^k \geq M$ then $y_{ij}^k = y_{ij}^k - M$ or if $y_{ij}^k < 0$ then $y_{ij}^k = M + y_{ij}^k$.

One consideration in PSO is that velocities may increase (or decrease) without bound, leading to particles making large steps in the solution space. Kennedy *et al.* [3] use a constant V_{max} to limit the range of velocity, requiring that $v_{ij}^k \in (-V_{max}, V_{max})$, $\forall i, j, k$. Setting V_{max} in this

application is informed by the following observations. First, if a particle velocity is near M , the bounceback and wrap-around solutions tend to be the same. Second, if the particle velocity is limited to too small of a value, the jobs may rearrange order on same machine many times before a job moves to a different machine. When we are using bounceback, only when y_{ij}^k is near $M/4$ and v_{ij}^k is near $M/2$, do we bounceback to the same machine and roughly the same location. When we wrap-around, the job will always end up on different machine. In order to balance these concerns, we use $V_{\max} = M/2$ in this study.

5 Computational experiments

The proposed PSO algorithm is implemented in C and tested on a large set of problem instances found in Kurz and Askin [4] and available from Kurz. The solution qualities are evaluated with the help of a lower bound developed by Kurz and Askin [4]. Based on Sankaran [6], we fix the PSO parameters as follows: $w=1$, $c_1=1.5$, $c_2=1.5$. The “bounceback” and “wrap-around” models are applied to each of the 1800 problem instances; the resultant 3600 algorithm- problem instance combinations are replicated 50 times with independent random numbers generated with the Mersenne Twister [5]. We terminate after 500 iterations, and the swarm size is 100 particles. The algorithms are run on a distributed computing environment called Condor in which heterogeneous Windows 7 machines are used; for this reason we do not report computing times.

We analyze the results based on the loss function, computed as the difference between the makespan and the lower bound, divided by the lower bound, averaged over 50 replications per algorithm and problem instance. The Kolmogorov-Smirnov test for normality indicates that neither the “bounceback” and “wrap-around” models produce results that are normally distributed, with p-values less than 0.01. A one-tailed Mann-Whitney test, with p-value less than 0.05, allows us to conclude that the “bounceback” model appears to produce better results.

6 Conclusions

Adapting PSO to a combinatorial optimization problem requires adaptations that are insightful regarding the how the velocity and location updates interaction with solution representation. We have illustrated that the adaptation impacts solution quality in a statistically significant manner. Future work will focus on comparing the random keys encoding with alternatives and larger scale computational experiments.

Acknowledgements We gratefully thank Edward Duffy, from CyberInfrastructure Technology Integration at Clemson University, for innumerable suggestions and immeasurable patience.

References

1. Eberhart, R.C. and Shi, Y., “Comparing inertia weights and constriction factors in particle swarm optimization”, *Proceedings of Congress on Evolutionary Computing*, 84-88, 2000.
2. Kennedy, J and Eberhart, R.C., “Particle swarm optimization”, *Proceedings of IEEE International Conference on Neural Networks*, 1942-1948, 1995.
3. Kennedy, J., Eberhart, R.C and Shi, Y. *Swarm Intelligence*. Morgan Kaufmann: CA, 2001.
4. Kurz, M.E. and Askin, R.G., “Scheduling flexible flow lines with sequence-dependent setup times”, *European Journal of Operational Research*, 159, 66-82, 2004.

5. Matsumoto, M. and Nishimura, T., "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation*, 8, 3-30, 1998.
6. Sankaran, V. *A Particle Swarm Optimization Using Random Keys for Flexible Flow Shop Scheduling Problem with Sequence Dependent Setup Times*, Master's Thesis, Clemson University, 2009.

Author Index

Adasme.....	445
Akkan	506
Almeder	557
Ascó	419
Ásgeirsson	308
Atakan	215
Atkin.....	146, 161, 419
Bai	161
Balasubramanian.....	568
Barnhart.....	352
Baumann P.....	539, 596
Baumann M.	542
Bäumelt	576
Bennell	486
Bertocchi	607
Bierwirth.....	536
Bispo	275
Borgulya	127
Borodin.....	469
Brandinu	137
Bülbül	215
Bürgy.....	553
Burke.....	146, 322, 335, 419
Buscher.....	519
Čapek.....	561
Charalambous	529
Cheng	408
Créput.....	633
Croce	240
Dauzère-Pérès	645
De Bruyn	469
De Causmaecker	374, 579
Dimitrov	542
Dobson	297
Donaldson.....	624
Dowland.....	584
Elliot.....	322
Fleszar	529
Fordyce.....	10
Fowler	408, 546, 568, 573, 599, 642
Francis	322
Gao	602
Glazebrook	94
Gonzales	261
Grinshpoun	173
Gröflin	553

Gui.....	497
Hajjam.....	633
Hanen	245
Hanzálek.....	245, 561, 576
Haspeslagh.....	579
Herring	587
Herrmann.....	200, 587
Hertzler.....	408
Hieu.....	493
Hodge.....	94
Hsieh	195
Huang.....	497
Huff.....	408
Hui.....	602
Ilani	173
Innorta	607
Kedad-Sidhoum	261
Keha	568
Kempf.....	408
Kendall	161
Khosravi	486
Khowala.....	568
Kirkbride	94
Koukam	633
Koulamas.....	240
Kozma	489
Kuhpfahl.....	536
Kurz.....	653
Kyngäs.....	308
Lenté.....	523
Leung.....	602
Li, J.....	161
Li, J-Q	484
Li X.	546
Li S.	633
Liedloff.....	523
Lin Y.K.....	195
Lin G.J.	195
Lisser.....	445
Lokhmotov	322
Lozano.....	564
Ma	497
Mackulak.....	599
Mannino.....	592
Mareček.....	322
Marla	352
Mason.....	642
McCollum.....	624
Medaglia.....	564
Milne	10
Ming.....	493

Minty.....	94
Mirchandani.....	484
Misir	374
Mönch	185, 519, 557, 642
Morrison	639
Müller	649
Ng.....	394
Noyan	215
Nurmi	308
Obeid.....	645
Pabst.....	599
Pais.....	335
Park	639
Parkes	322
Pazienza.....	489
Pfund	573
Pillay	628
Potts.....	486
Quadrifoglio.....	502
Rainwater.....	642
Ramacher	185
Ravizza	146
Riise	592
Rizzo	366
Romeijnnders.....	637
Rose.....	459
Rudová	649
Saner.....	596
Sankaran	653
Schäfer.....	596
Schenk.....	542
Scott	624
Seddik.....	261
Shen C-W.	502
Shen L.	519
Shufan	173
Sierksma	637
Silva-Lugo	489
Smet	549
Soto	445
Soukhal.....	523
Steinhöfel.....	297
Stølevik.....	308
Šůcha	532, 561, 576
Sun	546
Sy	394
Tezel.....	215
Thompson.....	584
T'Kindt.....	523
Trautmann.....	137, 539, 596
Uhlig.....	459

Urrutia	366
Vakhania.....	230
Van Vreckem	469
Vancroonenburg.....	549
Vanden Berghe	374, 549
Verbeeck.....	374
Vespucci	607
Wallace.....	602, 605
Werbos	489
Whitley	113
Xie.....	497
Yugma	645
Zajiček.....	532
Zhao Y.....	605
Zigrino.....	607